27 Input/output library [input.output]

27.1 General [input.output.general]

1 This Clause describes components that C++ programs may use to perform input/output operations.

2 The following subclauses describe requirements for stream parameters, and components for forward declarations of iostreams, predefined iostreams objects, base iostreams classes, stream buffering, stream formatting and manipulators, string streams, and file streams, as summarized in Table 121.

Table 121 — Input/output library summary

	Subclause	Header(s)
27.2	Requirements	
27.3	Forward declarations	`<iosfwd>`
27.4	Standard iostream objects	`<iostream>`
27.5	Iostreams base classes	`<ios>`
27.6	Stream buffers	`<streambuf>`

27.7	Formatting and manipulators	`<istream>`
		`<ostream>`
		`<iomanip>`

27.8	String streams	`<sstream>`

27.9	File streams	`<fstream>`
		`<cstdio>`
		`<cinttypes>`

3 Figure 7 illustrates relationships among various types described in this clause. A line from **A** to **B** indicates that **A** is an alias (e.g. a typedef) for **B** or that **A** is defined in terms of **B**.

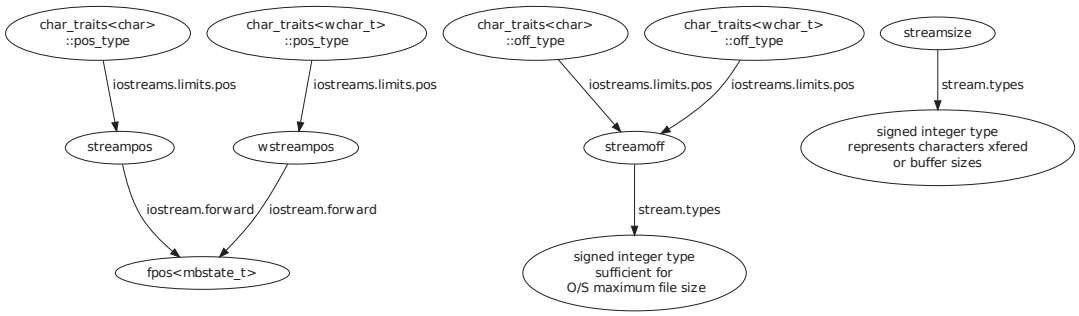

Figure 7 — Stream position, offset, and size types [non-normative]

27.2 Iostreams requirements [iostreams.requirements]

27.2.1 Imbue limitations [iostream.limits.imbue]

1 No function described in Clause 27 except for `ios_base::imbue` and `basic_filebuf::pubimbue` causes any

instance of `basic_ios::imbue` or `basic_streambuf::imbue` to be called. If any user function called from a function declared in Clause 27 or as an overriding virtual function of any class declared in Clause 27 calls `imbue`, the behavior is undefined.

27.2.2 Positioning type limitations [iostreams.limits.pos]

1 The classes of Clause 27 with template arguments `charT` and `traits` behave as described if `traits::pos_type` and `traits::off_type` are `streampos` and `streamoff` respectively. Except as noted explicitly below, their behavior when `traits::pos_type` and `traits::off_type` are other types is implementation-defined.

2 In the classes of Clause 27, a template formal parameter with name `charT` represents a member of the set of types containing `char`, `wchar_t`, and any other implementation-defined character types that satisfy the requirements for a character on which any of the iostream components can be instantiated.

27.2.3 Thread safety [iostreams.threadsafety]

1 Concurrent access to a stream object (27.8, 27.9), stream buffer object (27.6), or C Library stream (27.9.2) by multiple threads may result in a data race (1.10) unless otherwise specified (27.4). [*Note:* Data races result in undefined behavior (1.10). — *end note*]

2 If one thread makes a library call a that writes a value to a stream and, as a result, another thread reads this value from the stream through a library call b such that this does not result in a data race, then a's write synchronizes with b's read.

27.3 Forward declarations [iostream.forward]

Header `<iosfwd>` synopsis

```
namespace std {
  template<class charT> class char_traits;
  template<> class char_traits<char>;
  template<> class char_traits<char16_t>;
  template<> class char_traits<char32_t>;
  template<> class char_traits<wchar_t>;

  template<class T> class allocator;

  template <class charT, class traits = char_traits<charT> >
    class basic_ios;
  template <class charT, class traits = char_traits<charT> >
    class basic_streambuf;
  template <class charT, class traits = char_traits<charT> >
    class basic_istream;
  template <class charT, class traits = char_traits<charT> >
    class basic_ostream;
  template <class charT, class traits = char_traits<charT> >
    class basic_iostream;

  template <class charT, class traits = char_traits<charT>,
      class Allocator = allocator<charT> >
    class basic_stringbuf;
  template <class charT, class traits = char_traits<charT>,
      class Allocator = allocator<charT> >
    class basic_istringstream;
  template <class charT, class traits = char_traits<charT>,
      class Allocator = allocator<charT> >
    class basic_ostringstream;
  template <class charT, class traits = char_traits<charT>,
      class Allocator = allocator<charT> >
    class basic_stringstream;
```



```
  template <class charT, class traits = char_traits<charT> >
    class basic_filebuf;
  template <class charT, class traits = char_traits<charT> >
    class basic_ifstream;
  template <class charT, class traits = char_traits<charT> >
    class basic_ofstream;
  template <class charT, class traits = char_traits<charT> >
    class basic_fstream;

  template <class charT, class traits = char_traits<charT> >
    class istreambuf_iterator;
  template <class charT, class traits = char_traits<charT> >
    class ostreambuf_iterator;

  typedef basic_ios<char>       ios;
  typedef basic_ios<wchar_t>    wios;

  typedef basic_streambuf<char> streambuf;
  typedef basic_istream<char>   istream;
  typedef basic_ostream<char>   ostream;
  typedef basic_iostream<char>  iostream;

  typedef basic_stringbuf<char>     stringbuf;
  typedef basic_istringstream<char> istringstream;
  typedef basic_ostringstream<char> ostringstream;
  typedef basic_stringstream<char>  stringstream;

  typedef basic_filebuf<char>  filebuf;
  typedef basic_ifstream<char> ifstream;
  typedef basic_ofstream<char> ofstream;
  typedef basic_fstream<char>  fstream;

  typedef basic_streambuf<wchar_t> wstreambuf;
  typedef basic_istream<wchar_t>   wistream;
  typedef basic_ostream<wchar_t>   wostream;
  typedef basic_iostream<wchar_t>  wiostream;

  typedef basic_stringbuf<wchar_t>     wstringbuf;
  typedef basic_istringstream<wchar_t> wistringstream;
  typedef basic_ostringstream<wchar_t> wostringstream;
  typedef basic_stringstream<wchar_t>  wstringstream;

  typedef basic_filebuf<wchar_t>  wfilebuf;
  typedef basic_ifstream<wchar_t> wifstream;
  typedef basic_ofstream<wchar_t> wofstream;
  typedef basic_fstream<wchar_t>  wfstream;

  template <class state> class fpos;
  typedef fpos<char_traits<char>::state_type>    streampos;
  typedef fpos<char_traits<wchar_t>::state_type> wstreampos;
}
```

1 Default template arguments are described as appearing both in `<iosfwd>` and in the synopsis of other headers but it is well-formed to include both `<iosfwd>` and one or more of the other headers.[293]

293) It is the implementation's responsibility to implement headers so that including `<iosfwd>` and other headers does not

2 [*Note:* The class template specialization `basic_ios<charT,traits>` serves as a virtual base class for the class templates `basic_istream`, `basic_ostream`, and class templates derived from them. `basic_iostream` is a class template derived from both `basic_istream<charT,traits>` and `basic_ostream<charT,traits>`.

3 The class template specialization `basic_streambuf<charT,traits>` serves as a base class for class templates `basic_stringbuf` and `basic_filebuf`.

4 The class template specialization `basic_istream<charT,traits>` serves as a base class for class templates `basic_istringstream` and `basic_ifstream`.

5 The class template specialization `basic_ostream<charT,traits>` serves as a base class for class templates `basic_ostringstream` and `basic_ofstream`.

6 The class template specialization `basic_iostream<charT,traits>` serves as a base class for class templates `basic_stringstream` and `basic_fstream`.

7 Other typedefs define instances of class templates specialized for `char` or `wchar_t` types.

8 Specializations of the class template `fpos` are used for specifying file position information.

9 The types `streampos` and `wstreampos` are used for positioning streams specialized on `char` and `wchar_t` respectively.

10 This synopsis suggests a circularity between `streampos` and `char_traits<char>`. An implementation can avoid this circularity by substituting equivalent types. One way to do this might be

```
template<class stateT> class fpos { ... };       // depends on nothing
typedef ... _STATE;              // implementation private declaration of stateT

typedef fpos<_STATE> streampos;

template<> struct char_traits<char> {
  typedef streampos
  pos_type;
}
```

— *end note*]

27.4 Standard iostream objects [iostream.objects]

27.4.1 Overview [iostream.objects.overview]

Header `<iostream>` synopsis

```
#include <ios>
#include <streambuf>
#include <istream>
#include <ostream>

namespace std {
  extern istream cin;
  extern ostream cout;
  extern ostream cerr;
  extern ostream clog;

  extern wistream wcin;
  extern wostream wcout;
  extern wostream wcerr;
  extern wostream wclog;
}
```

1 The header `<iostream>` declares objects that associate objects with the standard C streams provided for by the functions declared in `<cstdio>` (27.9.2), and includes all the headers necessary to use these objects.

violate the rules about multiple occurrences of default arguments.

2 The objects are constructed and the associations are established at some time prior to or during the first time an object of class `ios_base::Init` is constructed, and in any case before the body of `main` begins execution.[294] The objects are not destroyed during program execution.[295] The results of including `<iostream>` in a translation unit shall be as if `<iostream>` defined an instance of `ios_base::Init` with static storage duration. Similarly, the entire program shall behave as if there were at least one instance of `ios_base::Init` with static storage duration.

3 Mixing operations on corresponding wide- and narrow-character streams follows the same semantics as mixing such operations on `FILE`s, as specified in Amendment 1 of the ISO C standard.

4 Concurrent access to a synchronized (27.5.3.4) standard iostream object's formatted and unformatted input (27.7.2.1) and output (27.7.3.1) functions or a standard C stream by multiple threads shall not result in a data race (1.10). [*Note:* Users must still synchronize concurrent use of these objects and streams by multiple threads if they wish to avoid interleaved characters. — *end note*]

27.4.2 Narrow stream objects [narrow.stream.objects]

`istream cin;`

1 The object `cin` controls input from a stream buffer associated with the object `stdin`, declared in `<cstdio>`.

2 After the object `cin` is initialized, `cin.tie()` returns `&cout`. Its state is otherwise the same as required for `basic_ios<char>::init` (27.5.5.2).

`ostream cout;`

3 The object `cout` controls output to a stream buffer associated with the object `stdout`, declared in `<cstdio>` (27.9.2).

`ostream cerr;`

4 The object `cerr` controls output to a stream buffer associated with the object `stderr`, declared in `<cstdio>` (27.9.2).

5 After the object `cerr` is initialized, `cerr.flags() & unitbuf` is nonzero and `cerr.tie()` returns `&cout`. Its state is otherwise the same as required for `basic_ios<char>::init` (27.5.5.2).

`ostream clog;`

6 The object `clog` controls output to a stream buffer associated with the object `stderr`, declared in `<cstdio>` (27.9.2).

27.4.3 Wide stream objects [wide.stream.objects]

`wistream wcin;`

1 The object `wcin` controls input from a stream buffer associated with the object `stdin`, declared in `<cstdio>`.

2 After the object `wcin` is initialized, `wcin.tie()` returns `&wcout`. Its state is otherwise the same as required for `basic_ios<wchar_t>::init` (27.5.5.2).

294) If it is possible for them to do so, implementations are encouraged to initialize the objects earlier than required.
295) Constructors and destructors for static objects can access these objects to read input from `stdin` or write output to `stdout` or `stderr`.


```
wostream wcout;
```

3 The object `wcout` controls output to a stream buffer associated with the object `stdout`, declared in `<cstdio>` (27.9.2).

```
wostream wcerr;
```

4 The object `wcerr` controls output to a stream buffer associated with the object `stderr`, declared in `<cstdio>` (27.9.2).

5 After the object `wcerr` is initialized, `wcerr.flags() & unitbuf` is nonzero and `wcerr.tie()` returns `&wcout`. Its state is otherwise the same as required for `basic_ios<wchar_t>::init` (27.5.5.2).

```
wostream wclog;
```

6 The object `wclog` controls output to a stream buffer associated with the object `stderr`, declared in `<cstdio>` (27.9.2).

27.5 Iostreams base classes [iostreams.base]

27.5.1 Overview [iostreams.base.overview]

Header `<ios>` synopsis

```
#include <iosfwd>

namespace std {
  typedef implementation-defined streamoff;
  typedef implementation-defined streamsize;
  template <class stateT> class fpos;

  class ios_base;
  template <class charT, class traits = char_traits<charT> >
    class basic_ios;

  // 27.5.6, manipulators:
  ios_base& boolalpha  (ios_base& str);
  ios_base& noboolalpha(ios_base& str);

  ios_base& showbase   (ios_base& str);
  ios_base& noshowbase (ios_base& str);

  ios_base& showpoint  (ios_base& str);
  ios_base& noshowpoint(ios_base& str);

  ios_base& showpos    (ios_base& str);
  ios_base& noshowpos  (ios_base& str);

  ios_base& skipws     (ios_base& str);
  ios_base& noskipws   (ios_base& str);

  ios_base& uppercase  (ios_base& str);
  ios_base& nouppercase(ios_base& str);

  ios_base& unitbuf    (ios_base& str);
```



```
    ios_base& nounitbuf  (ios_base& str);

    // 27.5.6.2 adjustfield:
    ios_base& internal   (ios_base& str);
    ios_base& left       (ios_base& str);
    ios_base& right      (ios_base& str);

    // 27.5.6.3 basefield:
    ios_base& dec        (ios_base& str);
    ios_base& hex        (ios_base& str);
    ios_base& oct        (ios_base& str);

    // 27.5.6.4 floatfield:
    ios_base& fixed      (ios_base& str);
    ios_base& scientific (ios_base& str);
    ios_base& hexfloat   (ios_base& str);
    ios_base& defaultfloat(ios_base& str);

    // 27.5.6.5 error reporting:
    enum class io_errc {
      stream = 1
    };

    template <> struct is_error_code_enum<io_errc> : public true_type { };
    error_code make_error_code(io_errc e) noexcept;
    error_condition make_error_condition(io_errc e) noexcept;
    const error_category& iostream_category() noexcept;
  }
```

27.5.2 Types [stream.types]

```
typedef implementation-defined streamoff;
```

1 The type `streamoff` is a synonym for one of the signed basic integral types of sufficient size to represent the maximum possible file size for the operating system.[296]

```
typedef implementation-defined streamsize;
```

2 The type `streamsize` is a synonym for one of the signed basic integral types. It is used to represent the number of characters transferred in an I/O operation, or the size of I/O buffers.[297]

27.5.3 Class `ios_base` [ios.base]

```
namespace std {
  class ios_base {
  public:
    class failure;

    // 27.5.3.1.2 fmtflags
    typedef T1 fmtflags;
    static constexpr fmtflags boolalpha = unspecified ;
```

296) Typically `long long`.

297) `streamsize` is used in most places where ISO C would use `size_t`. Most of the uses of `streamsize` could use `size_t`, except for the `strstreambuf` constructors, which require negative values. It should probably be the signed type corresponding to `size_t` (which is what Posix.2 calls `ssize_t`).


```
static constexpr fmtflags dec = unspecified ;
static constexpr fmtflags fixed = unspecified ;
static constexpr fmtflags hex = unspecified ;
static constexpr fmtflags internal = unspecified ;
static constexpr fmtflags left = unspecified ;
static constexpr fmtflags oct = unspecified ;
static constexpr fmtflags right = unspecified ;
static constexpr fmtflags scientific = unspecified ;
static constexpr fmtflags showbase = unspecified ;
static constexpr fmtflags showpoint = unspecified ;
static constexpr fmtflags showpos = unspecified ;
static constexpr fmtflags skipws = unspecified ;
static constexpr fmtflags unitbuf = unspecified ;
static constexpr fmtflags uppercase = unspecified ;
static constexpr fmtflags adjustfield = see below;
static constexpr fmtflags basefield = see below;
static constexpr fmtflags floatfield = see below;

// 27.5.3.1.3 iostate
typedef T2 iostate;
static constexpr iostate badbit = unspecified ;
static constexpr iostate eofbit = unspecified ;
static constexpr iostate failbit = unspecified ;
static constexpr iostate goodbit = see below;

// 27.5.3.1.4 openmode
typedef T3 openmode;
static constexpr openmode app = unspecified ;
static constexpr openmode ate = unspecified ;
static constexpr openmode binary = unspecified ;
static constexpr openmode in = unspecified ;
static constexpr openmode out = unspecified ;
static constexpr openmode trunc = unspecified ;

// 27.5.3.1.5 seekdir
typedef T4 seekdir;
static constexpr seekdir beg = unspecified ;
static constexpr seekdir cur = unspecified ;
static constexpr seekdir end = unspecified ;

class Init;

// 27.5.3.2 fmtflags state:
fmtflags flags() const;
fmtflags flags(fmtflags fmtfl);
fmtflags setf(fmtflags fmtfl);
fmtflags setf(fmtflags fmtfl, fmtflags mask);
void unsetf(fmtflags mask);

streamsize precision() const;
streamsize precision(streamsize prec);
streamsize width() const;
streamsize width(streamsize wide);

// 27.5.3.3 locales:
```



```
    locale imbue(const locale& loc);
    locale getloc() const;

    // 27.5.3.5 storage:
    static int xalloc();
    long&  iword(int index);
    void*& pword(int index);

    // destructor
    virtual ~ios_base();

    // 27.5.3.6 callbacks;
    enum event { erase_event, imbue_event, copyfmt_event };
    typedef void (*event_callback)(event, ios_base&, int index);
    void register_callback(event_callback fn, int index);

    ios_base(const ios_base&) = delete;
    ios_base& operator=(const ios_base&) = delete;

    static bool sync_with_stdio(bool sync = true);

  protected:
    ios_base();

  private:
    static int index;  // exposition only
    long* iarray;      // exposition only
    void** parray;     // exposition only
  };
}
```

1 `ios_base` defines several member types:

— a class `failure` derived from `system_error`;

— a class `Init`;

— three bitmask types, `fmtflags`, `iostate`, and `openmode`;

— an enumerated type, `seekdir`.

2 It maintains several kinds of data:

— state information that reflects the integrity of the stream buffer;

— control information that influences how to interpret (format) input sequences and how to generate (format) output sequences;

— additional information that is stored by the program for its private use.

3 [*Note:* For the sake of exposition, the maintained data is presented here as:

— `static int index`, specifies the next available unique index for the integer or pointer arrays maintained for the private use of the program, initialized to an unspecified value;

— `long* iarray`, points to the first element of an arbitrary-length `long` array maintained for the private use of the program;

— `void** parray`, points to the first element of an arbitrary-length pointer array maintained for the private use of the program. — *end note*]

27.5.3.1 Types [ios.types]

27.5.3.1.1 Class `ios_base::failure` [ios::failure]

```
namespace std {
  class ios_base::failure : public system_error {
  public:
    explicit failure(const string& msg, const error_code& ec = io_errc::stream);
    explicit failure(const char* msg, const error_code& ec = io_errc::stream);
  };
}
```

1 The class `failure` defines the base class for the types of all objects thrown as exceptions, by functions in the iostreams library, to report errors detected during stream buffer operations.

2 When throwing `ios_base::failure` exceptions, implementations should provide values of `ec` that identify the specific reason for the failure. [*Note:* Errors arising from the operating system would typically be reported as `system_category()` errors with an error value of the error number reported by the operating system. Errors arising from within the stream library would typically be reported as `error_code(io_errc::stream, iostream_category())`. — *end note*]

```
explicit failure(const string& msg, const error_code& ec = io_errc::stream);
```

3 *Effects:* Constructs an object of class `failure` by constructing the base class with `msg` and `ec`.

```
explicit failure(const char* msg, const error_code& ec = io_errc::stream);
```

4 *Effects:* Constructs an object of class `failure` by constructing the base class with `msg` and `ec`.

27.5.3.1.2 Type `ios_base::fmtflags` [ios::fmtflags]

```
typedef T1 fmtflags;
```

1 The type `fmtflags` is a bitmask type (17.5.2.1.3). Setting its elements has the effects indicated in Table 122.

2 Type `fmtflags` also defines the constants indicated in Table 123.

27.5.3.1.3 Type `ios_base::iostate` [ios::iostate]

```
typedef T2 iostate;
```

1 The type `iostate` is a bitmask type (17.5.2.1.3) that contains the elements indicated in Table 124.

2 Type `iostate` also defines the constant:

— `goodbit`, the value zero.

27.5.3.1.4 Type `ios_base::openmode` [ios::openmode]

```
typedef T3 openmode;
```

1 The type `openmode` is a bitmask type (17.5.2.1.3). It contains the elements indicated in Table 125.

27.5.3.1.5 Type `ios_base::seekdir` [ios::seekdir]

```
typedef T4 seekdir;
```

1 The type `seekdir` is an enumerated type (17.5.2.1.2) that contains the elements indicated in Table 126.

Table 122 — `fmtflags` effects

Element	Effect(s) if set
`boolalpha`	insert and extract `bool` type in alphabetic format
`dec`	converts integer input or generates integer output in decimal base
`fixed`	generate floating-point output in fixed-point notation
`hex`	converts integer input or generates integer output in hexadecimal base
`internal`	adds fill characters at a designated internal point in certain generated output, or identical to `right` if no such point is designated
`left`	adds fill characters on the right (final positions) of certain generated output
`oct`	converts integer input or generates integer output in octal base
`right`	adds fill characters on the left (initial positions) of certain generated output
`scientific`	generates floating-point output in scientific notation
`showbase`	generates a prefix indicating the numeric base of generated integer output
`showpoint`	generates a decimal-point character unconditionally in generated floating-point output
`showpos`	generates a `+` sign in non-negative generated numeric output
`skipws`	skips leading whitespace before certain input operations
`unitbuf`	flushes output after each output operation
`uppercase`	replaces certain lowercase letters with their uppercase equivalents in generated output

Table 123 — `fmtflags` constants

Constant	Allowable values
`adjustfield`	`left | right | internal`
`basefield`	`dec | oct | hex`
`floatfield`	`scientific | fixed`

Table 124 — `iostate` effects

Element	Effect(s) if set
`badbit`	indicates a loss of integrity in an input or output sequence (such as an irrecoverable read error from a file);
`eofbit`	indicates that an input operation reached the end of an input sequence;
`failbit`	indicates that an input operation failed to read the expected characters, or that an output operation failed to generate the desired characters.

Table 125 — `openmode` effects

Element	Effect(s) if set
`app`	seek to end before each write
`ate`	open and seek to end immediately after opening
`binary`	perform input and output in binary mode (as opposed to text mode)
`in`	open for input
`out`	open for output
`trunc`	truncate an existing stream when opening

Table 126 — `seekdir` effects

Element	Meaning
`beg`	request a seek (for subsequent input or output) relative to the beginning of the stream
`cur`	request a seek relative to the current position within the sequence
`end`	request a seek relative to the current end of the sequence

27.5.3.1.6 Class `ios_base::Init` [ios::Init]

```
namespace std {
  class ios_base::Init {
  public:
    Init();
   ~Init();
  private:
    static int init_cnt; // exposition only
  };
}
```

1 The class `Init` describes an object whose construction ensures the construction of the eight objects declared in `<iostream>` (27.4) that associate file stream buffers with the standard C streams provided for by the functions declared in `<cstdio>` (27.9.2).

2 For the sake of exposition, the maintained data is presented here as:

— `static int init_cnt`, counts the number of constructor and destructor calls for class `Init`, initialized to zero.

```
Init();
```

3 *Effects:* Constructs an object of class `Init`. Constructs and initializes the objects `cin`, `cout`, `cerr`, `clog`, `wcin`, `wcout`, `wcerr`, and `wclog` if they have not already been constructed and initialized.

```
~Init();
```

4 *Effects:* Destroys an object of class `Init`. If there are no other instances of the class still in existence, calls `cout.flush()`, `cerr.flush()`, `clog.flush()`, `wcout.flush()`, `wcerr.flush()`, `wclog.flush()`.

27.5.3.2 `ios_base` state functions [fmtflags.state]

```
fmtflags flags() const;
```

1 *Returns:* The format control information for both input and output.

```
fmtflags flags(fmtflags fmtfl);
```

2 *Postcondition:* `fmtfl == flags()`.

3 *Returns:* The previous value of `flags()`.

```
fmtflags setf(fmtflags fmtfl);
```


4 *Effects:* Sets `fmtfl` in `flags()`.

5 *Returns:* The previous value of `flags()`.

```
fmtflags setf(fmtflags fmtfl, fmtflags mask);
```

6 *Effects:* Clears `mask` in `flags()`, sets `fmtfl & mask` in `flags()`.

7 *Returns:* The previous value of `flags()`.

```
void unsetf(fmtflags mask);
```

8 *Effects:* Clears `mask` in `flags()`.

```
streamsize precision() const;
```

9 *Returns:* The precision to generate on certain output conversions.

```
streamsize precision(streamsize prec);
```

10 *Postcondition:* `prec == precision()`.

11 *Returns:* The previous value of `precision()`.

```
streamsize width() const;
```

12 *Returns:* The minimum field width (number of characters) to generate on certain output conversions.

```
streamsize width(streamsize wide);
```

13 *Postcondition:* `wide == width()`.

14 *Returns:* The previous value of `width()`.

27.5.3.3 `ios_base` functions [ios.base.locales]

```
locale imbue(const locale& loc);
```

1 *Effects:* Calls each registered callback pair `(fn,index)` (27.5.3.6) as `(*fn)(imbue_event,*this,index)` at such a time that a call to `ios_base::getloc()` from within `fn` returns the new locale value `loc`.

2 *Returns:* The previous value of `getloc()`.

3 *Postcondition:* `loc == getloc()`.

```
locale getloc() const;
```

4 *Returns:* If no locale has been imbued, a copy of the global C++ locale, `locale()`, in effect at the time of construction. Otherwise, returns the imbued locale, to be used to perform locale-dependent input and output operations.

27.5.3.4 `ios_base` static members [ios.members.static]

```
bool sync_with_stdio(bool sync = true);
```

1 *Returns:* `true` if the previous state of the standard iostream objects (27.4) was synchronized and otherwise returns `false`. The first time it is called, the function returns `true`.

2 *Effects:* If any input or output operation has occurred using the standard streams prior to the call, the effect is implementation-defined. Otherwise, called with a false argument, it allows the standard streams to operate independently of the standard C streams.

3 When a standard iostream object `str` is *synchronized* with a standard stdio stream `f`, the effect of inserting a character `c` by

```
fputc(f, c);
```

is the same as the effect of

```
str.rdbuf()->sputc(c);
```

for any sequences of characters; the effect of extracting a character `c` by

```
c = fgetc(f);
```

is the same as the effect of

```
c = str.rdbuf()->sbumpc();
```

for any sequences of characters; and the effect of pushing back a character `c` by

```
ungetc(c, f);
```

is the same as the effect of

```
str.rdbuf()->sputbackc(c);
```

for any sequence of characters.[298]

27.5.3.5 `ios_base` storage functions [ios.base.storage]

```
static int xalloc();
```

1 *Returns:* `index ++`.

2 *Remarks:* Concurrent access to this function by multiple threads shall not result in a data race (1.10).

```
long& iword(int idx);
```

3 *Effects:* If `iarray` is a null pointer, allocates an array of `long` of unspecified size and stores a pointer to its first element in `iarray`. The function then extends the array pointed at by `iarray` as necessary to include the element `iarray[idx]`. Each newly allocated element of the array is initialized to zero. The reference returned is invalid after any other operations on the object.[299] However, the value of the storage referred to is retained, so that until the next call to `copyfmt`, calling `iword` with the same index yields another reference to the same value. If the function fails[300] and `*this` is a base subobject of a

298) This implies that operations on a standard iostream object can be mixed arbitrarily with operations on the corresponding stdio stream. In practical terms, synchronization usually means that a standard iostream object and a standard stdio object share a buffer.

299) An implementation is free to implement both the integer array pointed at by `iarray` and the pointer array pointed at by `parray` as sparse data structures, possibly with a one-element cache for each.

300) for example, because it cannot allocate space.

`basic_ios<>` object or subobject, the effect is equivalent to calling `basic_ios<>::setstate(badbit)` on the derived object (which may throw `failure`).

4 *Returns:* On success `iarray[idx]`. On failure, a valid `long&` initialized to 0.

```
void*& pword(int idx);
```

5 *Effects:* If `parray` is a null pointer, allocates an array of pointers to `void` of unspecified size and stores a pointer to its first element in `parray`. The function then extends the array pointed at by `parray` as necessary to include the element `parray[idx]`. Each newly allocated element of the array is initialized to a null pointer. The reference returned is invalid after any other operations on the object. However, the value of the storage referred to is retained, so that until the next call to `copyfmt`, calling `pword` with the same index yields another reference to the same value. If the function fails[301] and `*this` is a base subobject of a `basic_ios<>` object or subobject, the effect is equivalent to calling `basic_ios<>::setstate(badbit)` on the derived object (which may throw `failure`).

6 *Returns:* On success `parray[idx]`. On failure a valid `void*&` initialized to 0.

7 *Remarks:* After a subsequent call to `pword(int)` for the same object, the earlier return value may no longer be valid.

27.5.3.6 `ios_base` callbacks [ios.base.callback]

```
void register_callback(event_callback fn, int index);
```

1 *Effects:* Registers the pair `(fn,index)` such that during calls to `imbue()` (27.5.3.3), `copyfmt()`, or `~ios_base()` (27.5.3.7), the function `fn` is called with argument `index`. Functions registered are called when an event occurs, in opposite order of registration. Functions registered while a callback function is active are not called until the next event.

2 *Requires:* The function `fn` shall not throw exceptions.

Remarks: Identical pairs are not merged. A function registered twice will be called twice.

27.5.3.7 `ios_base` constructors/destructor [ios.base.cons]

```
ios_base();
```

1 *Effects:* Each `ios_base` member has an indeterminate value after construction. The object's members shall be initialized by calling `basic_ios::init` before the object's first use or before it is destroyed, whichever comes first; otherwise the behavior is undefined.

```
~ios_base()
```

2 *Effects:* Destroys an object of class `ios_base`. Calls each registered callback pair `(fn, index)` (27.5.3.6) as `(*fn)(erase_event, *this, index)` at such time that any `ios_base` member function called from within `fn` has well defined results.

27.5.4 Class template `fpos` [fpos]

```
namespace std {
  template <class stateT> class fpos {
  public:
    // 27.5.4.1 Members
    stateT state() const;
```

301) for example, because it cannot allocate space.


```
    void state(stateT);
  private;
    stateT st; // exposition only
  };
}
```

27.5.4.1 `fpos` members [fpos.members]

```
void state(stateT s);
```

1 *Effects:* Assign `s` to `st`.

```
stateT state() const;
```

2 *Returns:* Current value of `st`.

27.5.4.2 `fpos` requirements [fpos.operations]

1 Operations specified in Table 127 are permitted. In that table,

— `P` refers to an instance of `fpos`,

— `p` and `q` refer to values of type `P`,

— `O` refers to type `streamoff`,

— `o` refers to a value of type `streamoff`,

— `sz` refers to a value of type `streamsize` and

— `i` refers to a value of type `int`.

Table 127 — Position type requirements

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
`P(i)`			`p == P(i)` note: a destructor is assumed.
`P p(i);` `P p = i;`			post: `p == P(i)`.
`P(o)`	`fpos`	converts from `offset`	
`O(p)`	streamoff	converts to `offset`	`P(O(p)) == p`
`p == q`	convertible to `bool`		`==` is an equivalence relation
`p != q`	convertible to `bool`	`!(p == q)`	
`q = p + o` `p += o`	`fpos`	`+` offset	`q - o == p`
`q = p - o` `p -= o`	`fpos`	`-` offset	`q + o == p`
`o = p - q`	streamoff	distance	`q + o == p`
`streamsize(o)` `O(sz)`	`streamsize` streamoff	converts converts	`streamsize(O(sz)) == sz` `streamsize(O(sz)) == sz`

2 [*Note:* Every implementation is required to supply overloaded operators on `fpos` objects to satisfy the requirements of 27.5.4.2. It is unspecified whether these operators are members of `fpos`, global operators, or provided in some other way. — *end note*]

3 Stream operations that return a value of type `traits::pos_type` return `P(O(-1))` as an invalid value to signal an error. If this value is used as an argument to any `istream`, `ostream`, or `streambuf` member that accepts a value of type `traits::pos_type` then the behavior of that function is undefined.

27.5.5 Class template `basic_ios` [ios]

27.5.5.1 Overview [ios.overview]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_ios : public ios_base {
  public:

    // types:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;

    explicit operator bool() const;
    bool operator!() const;
    iostate rdstate() const;
    void clear(iostate state = goodbit);
    void setstate(iostate state);
    bool good() const;
    bool eof()  const;
    bool fail() const;
    bool bad()  const;

    iostate exceptions() const;
    void exceptions(iostate except);

    // 27.5.5.2 Constructor/destructor:
    explicit basic_ios(basic_streambuf<charT,traits>* sb);
    virtual ~basic_ios();

    // 27.5.5.3 Members:
    basic_ostream<charT,traits>* tie() const;
    basic_ostream<charT,traits>* tie(basic_ostream<charT,traits>* tiestr);

    basic_streambuf<charT,traits>* rdbuf() const;
    basic_streambuf<charT,traits>* rdbuf(basic_streambuf<charT,traits>* sb);

    basic_ios& copyfmt(const basic_ios& rhs);

    char_type fill() const;
    char_type fill(char_type ch);

    locale imbue(const locale& loc);

    char     narrow(char_type c, char dfault) const;
    char_type widen(char c) const;

    basic_ios(const basic_ios& ) = delete;
    basic_ios& operator=(const basic_ios&) = delete;
```



```
  protected:
    basic_ios();
    void init(basic_streambuf<charT,traits>* sb);
    void move(basic_ios& rhs);
    void move(basic_ios&& rhs);
    void swap(basic_ios& rhs) noexcept;
    void set_rdbuf(basic_streambuf<charT, traits>* sb);

  };
}
```

27.5.5.2 `basic_ios` constructors [basic.ios.cons]

```
explicit basic_ios(basic_streambuf<charT,traits>* sb);
```

1 *Effects:* Constructs an object of class `basic_ios`, assigning initial values to its member objects by calling `init(sb)`.

```
basic_ios();
```

2 *Effects:* Constructs an object of class `basic_ios` (27.5.3.7) leaving its member objects uninitialized. The object shall be initialized by calling `basic_ios::init` before its first use or before it is destroyed, whichever comes first; otherwise the behavior is undefined.

```
~basic_ios();
```

3 *Remarks:* The destructor does not destroy `rdbuf()`.

```
void init(basic_streambuf<charT,traits>* sb);
```

Postconditions: The postconditions of this function are indicated in Table 128.

Table 128 — `basic_ios::init()` effects

Element	Value
`rdbuf()`	`sb`
`tie()`	`0`
`rdstate()`	`goodbit` if `sb` is not a null pointer, otherwise `badbit`.
`exceptions()`	`goodbit`
`flags()`	`skipws | dec`
`width()`	`0`
`precision()`	`6`
`fill()`	`widen(' ');`
`getloc()`	a copy of the value returned by `locale()`
iarray	a null pointer
parray	a null pointer

27.5.5.3 Member functions [basic.ios.members]

```
basic_ostream<charT,traits>* tie() const;
```

1 *Returns:* An output sequence that is *tied* to (synchronized with) the sequence controlled by the stream buffer.

```
basic_ostream<charT,traits>* tie(basic_ostream<charT,traits>* tiestr);
```

2 *Requires:* If `tiestr` is not null, `tiestr` must not be reachable by traversing the linked list of tied stream objects starting from `tiestr->tie()`.

3 *Postcondition:* `tiestr == tie()`.

4 *Returns:* The previous value of `tie()`.

```
basic_streambuf<charT,traits>* rdbuf() const;
```

5 *Returns:* A pointer to the `streambuf` associated with the stream.

```
basic_streambuf<charT,traits>* rdbuf(basic_streambuf<charT,traits>* sb);
```

6 *Postcondition:* `sb == rdbuf()`.

7 *Effects:* Calls `clear()`.

8 *Returns:* The previous value of `rdbuf()`.

```
locale imbue(const locale& loc);
```

9 *Effects:* Calls `ios_base::imbue(loc)` (27.5.3.3) and if `rdbuf()!=0` then `rdbuf()->pubimbue(loc)` (27.6.3.2.1).

10 *Returns:* The prior value of `ios_base::imbue()`.

```
char narrow(char_type c, char dfault) const;
```

11 *Returns:* `use_facet< ctype<char_type> >(getloc()).narrow(c,dfault)`

```
char_type widen(char c) const;
```

12 *Returns:* `use_facet< ctype<char_type> >(getloc()).widen(c)`

```
char_type fill() const;
```

13 *Returns:* The character used to pad (fill) an output conversion to the specified field width.

```
char_type fill(char_type fillch);
```

14 *Postcondition:* `traits::eq(fillch, fill())`

15 *Returns:* The previous value of `fill()`.


```
basic_ios& copyfmt(const basic_ios& rhs);
```

16 *Effects:* If `(this == &rhs)` does nothing. Otherwise assigns to the member objects of `*this` the corresponding member objects of `rhs` as follows:

1. calls each registered callback pair `(fn, index)` as `(*fn)(erase_event, *this, index)`;
2. assigns to the member objects of `*this` the corresponding member objects of `rhs`, except that
   - `rdstate()`, `rdbuf()`, and `exceptions()` are left unchanged;
   - the contents of arrays pointed at by `pword` and `iword` are copied, not the pointers themselves;[302] and
   - if any newly stored pointer values in `*this` point at objects stored outside the object `rhs` and those objects are destroyed when `rhs` is destroyed, the newly stored pointer values are altered to point at newly constructed copies of the objects;
3. calls each callback pair that was copied from `rhs` as `(*fn)(copyfmt_event, *this, index)`;
4. calls `exceptions(rhs.except())`.

17 *Note:* The second pass through the callback pairs permits a copied `pword` value to be zeroed, or to have its referent deep copied or reference counted, or to have other special action taken.

18 *Postconditions:* The postconditions of this function are indicated in Table 129.

Table 129 — `basic_ios::copyfmt()` effects

Element	Value
`rdbuf()`	*unchanged*
`tie()`	`rhs.tie()`
`rdstate()`	*unchanged*
`exceptions()`	`rhs.exceptions()`
`flags()`	`rhs.flags()`
`width()`	`rhs.width()`
`precision()`	`rhs.precision()`
`fill()`	`rhs.fill()`
`getloc()`	`rhs.getloc()`

19 *Returns:* `*this`.

```
void move(basic_ios& rhs);
void move(basic_ios&& rhs);
```

20 *Postconditions:* `*this` shall have the state that `rhs` had before the function call, except that `rdbuf()` shall return 0. `rhs` shall be in a valid but unspecified state, except that `rhs.rdbuf()` shall return the same value as it returned before the function call, and `rhs.tie()` shall return 0.

```
void swap(basic_ios& rhs) noexcept;
```

302) This suggests an infinite amount of copying, but the implementation can keep track of the maximum element of the arrays that is non-zero.

21 *Effects:* The states of `*this` and `rhs` shall be exchanged, except that `rdbuf()` shall return the same value as it returned before the function call, and `rhs.rdbuf()` shall return the same value as it returned before the function call.

```
void set_rdbuf(basic_streambuf<charT, traits>* sb);
```

22 *Requires:* `sb != nullptr`.

23 *Effects:* Associates the `basic_streambuf` object pointed to by `sb` with this stream without calling `clear()`.

24 *Postconditions:* `rdbuf() == sb`.

25 *Throws:* Nothing.

27.5.5.4 `basic_ios` flags functions [iostate.flags]

```
explicit operator bool() const;
```

1 *Returns:* `!fail()`.

```
bool operator!() const;
```

2 *Returns:* `fail()`.

```
iostate rdstate() const;
```

3 *Returns:* The error state of the stream buffer.

```
void clear(iostate state = goodbit);
```

4 *Postcondition:* If `rdbuf()!=0` then `state == rdstate()`; otherwise `rdstate()==(state | ios_base::badbit)`.

5 *Effects:* If `((state | (rdbuf() ? goodbit : badbit)) & exceptions()) == 0`, returns. Otherwise, the function throws an object `fail` of class `basic_ios::failure` (27.5.3.1.1), constructed with implementation-defined argument values.

```
void setstate(iostate state);
```

6 *Effects:* Calls `clear(rdstate() | state)` (which may throw `basic_ios::failure` (27.5.3.1.1)).

```
bool good() const;
```

7 *Returns:* `rdstate() == 0`

```
bool eof() const;
```

8 *Returns:* `true` if `eofbit` is set in `rdstate()`.

```
bool fail() const;
```


9 *Returns:* `true` if `failbit` or `badbit` is set in `rdstate()`.[303]

```
bool bad() const;
```

10 *Returns:* `true` if `badbit` is set in `rdstate()`.

```
iostate exceptions() const;
```

11 *Returns:* A mask that determines what elements set in `rdstate()` cause exceptions to be thrown.

```
void exceptions(iostate except);
```

12 *Postcondition:* `except == exceptions()`.

13 *Effects:* Calls `clear(rdstate())`.

27.5.6 `ios_base` manipulators [std.ios.manip]

27.5.6.1 `fmtflags` manipulators [fmtflags.manip]

```
ios_base& boolalpha(ios_base& str);
```

1 *Effects:* Calls `str.setf(ios_base::boolalpha)`.

2 *Returns:* `str`.

```
ios_base& noboolalpha(ios_base& str);
```

3 *Effects:* Calls `str.unsetf(ios_base::boolalpha)`.

4 *Returns:* `str`.

```
ios_base& showbase(ios_base& str);
```

5 *Effects:* Calls `str.setf(ios_base::showbase)`.

6 *Returns:* `str`.

```
ios_base& noshowbase(ios_base& str);
```

7 *Effects:* Calls `str.unsetf(ios_base::showbase)`.

8 *Returns:* `str`.

```
ios_base& showpoint(ios_base& str);
```

9 *Effects:* Calls `str.setf(ios_base::showpoint)`.

10 *Returns:* `str`.

```
ios_base& noshowpoint(ios_base& str);
```

303) Checking `badbit` also for `fail()` is historical practice.

11 *Effects:* Calls `str.unsetf(ios_base::showpoint)`.
12 *Returns:* `str`.

```
ios_base& showpos(ios_base& str);
```

13 *Effects:* Calls `str.setf(ios_base::showpos)`.
14 *Returns:* `str`.

```
ios_base& noshowpos(ios_base& str);
```

15 *Effects:* Calls `str.unsetf(ios_base::showpos)`.
16 *Returns:* `str`.

```
ios_base& skipws(ios_base& str);
```

17 *Effects:* Calls `str.setf(ios_base::skipws)`.
18 *Returns:* `str`.

```
ios_base& noskipws(ios_base& str);
```

19 *Effects:* Calls `str.unsetf(ios_base::skipws)`.
20 *Returns:* `str`.

```
ios_base& uppercase(ios_base& str);
```

21 *Effects:* Calls `str.setf(ios_base::uppercase)`.
22 *Returns:* `str`.

```
ios_base& nouppercase(ios_base& str);
```

23 *Effects:* Calls `str.unsetf(ios_base::uppercase)`.
24 *Returns:* `str`.

```
ios_base& unitbuf(ios_base& str);
```

25 *Effects:* Calls `str.setf(ios_base::unitbuf)`.
26 *Returns:* `str`.

```
ios_base& nounitbuf(ios_base& str);
```

27 *Effects:* Calls `str.unsetf(ios_base::unitbuf)`.
28 *Returns:* `str`.

27.5.6.2 `adjustfield` manipulators [adjustfield.manip]

```
ios_base& internal(ios_base& str);
```

1 *Effects:* Calls `str.setf(ios_base::internal, ios_base::adjustfield)`.

2 *Returns:* `str`.

```
ios_base& left(ios_base& str);
```

3 *Effects:* Calls `str.setf(ios_base::left, ios_base::adjustfield)`.

4 *Returns:* `str`.

```
ios_base& right(ios_base& str);
```

5 *Effects:* Calls `str.setf(ios_base::right, ios_base::adjustfield)`.

6 *Returns:* `str`.

27.5.6.3 `basefield` manipulators [basefield.manip]

```
ios_base& dec(ios_base& str);
```

1 *Effects:* Calls `str.setf(ios_base::dec, ios_base::basefield)`.

2 *Returns:* `str`[304].

```
ios_base& hex(ios_base& str);
```

3 *Effects:* Calls `str.setf(ios_base::hex, ios_base::basefield)`.

4 *Returns:* `str`.

```
ios_base& oct(ios_base& str);
```

5 *Effects:* Calls `str.setf(ios_base::oct, ios_base::basefield)`.

6 *Returns:* `str`.

27.5.6.4 `floatfield` manipulators [floatfield.manip]

```
ios_base& fixed(ios_base& str);
```

1 *Effects:* Calls `str.setf(ios_base::fixed, ios_base::floatfield)`.

2 *Returns:* `str`.

```
ios_base& scientific(ios_base& str);
```

3 *Effects:* Calls `str.setf(ios_base::scientific, ios_base::floatfield)`.

4 *Returns:* `str`.

```
ios_base& hexfloat(ios_base& str);
```

304) The function signature `dec(ios_base&)` can be called by the function signature `basic_ostream& stream::operator<<(ios_base& (*)(ios_base&))` to permit expressions of the form `cout <<dec` to change the format flags stored in `cout`.

5 *Effects:* Calls `str.setf(ios_base::fixed | ios_base::scientific, ios_base::floatfield)`.

6 *Returns:* `str`.

7 [*Note:* The more obvious use of `ios_base::hex` to specify hexadecimal floating-point format would change the meaning of existing well defined programs. C++2003 gives no meaning to the combination of `fixed` and `scientific`. — *end note*]

```
ios_base& defaultfloat(ios_base& str);
```

8 *Effects:* Calls `str.unsetf(ios_base::floatfield)`.

9 *Returns:* `str`.

27.5.6.5 Error reporting [error.reporting]

```
error_code make_error_code(io_errc e) noexcept;
```

1 *Returns:* `error_code(static_cast<int>(e), iostream_category())`.

```
error_condition make_error_condition(io_errc e) noexcept;
```

2 *Returns:* `error_condition(static_cast<int>(e), iostream_category())`.

```
const error_category& iostream_category() noexcept;
```

3 *Returns:* A reference to an object of a type derived from class `error_category`.

4 The object's `default_error_condition` and `equivalent` virtual functions shall behave as specified for the class `error_category`. The object's `name` virtual function shall return a pointer to the string `"iostream"`.

27.6 Stream buffers [stream.buffers]

27.6.1 Overview [stream.buffers.overview]

Header `<streambuf>` synopsis

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
    class basic_streambuf;
  typedef basic_streambuf<char>     streambuf;
  typedef basic_streambuf<wchar_t> wstreambuf;
}
```

1 The header `<streambuf>` defines types that control input from and output to *character* sequences.

27.6.2 Stream buffer requirements [streambuf.reqts]

1 Stream buffers can impose various constraints on the sequences they control. Some constraints are:

— The controlled input sequence can be not readable.

— The controlled output sequence can be not writable.

— The controlled sequences can be associated with the contents of other representations for character sequences, such as external files.

— The controlled sequences can support operations *directly* to or from associated sequences.

— The controlled sequences can impose limitations on how the program can read characters from a sequence, write characters to a sequence, put characters back into an input sequence, or alter the stream position.

2 Each sequence is characterized by three pointers which, if non-null, all point into the same `charT` array object. The array object represents, at any moment, a (sub)sequence of characters from the sequence. Operations performed on a sequence alter the values stored in these pointers, perform reads and writes directly to or from associated sequences, and alter "the stream position" and conversion state as needed to maintain this subsequence relationship. The three pointers are:

— the *beginning pointer*, or lowest element address in the array (called `xbeg` here);

— the *next pointer*, or next element address that is a current candidate for reading or writing (called `xnext` here);

— the *end pointer*, or first element address beyond the end of the array (called `xend` here).

3 The following semantic constraints shall always apply for any set of three pointers for a sequence, using the pointer names given immediately above:

— If `xnext` is not a null pointer, then `xbeg` and `xend` shall also be non-null pointers into the same `charT` array, as described above; otherwise, `xbeg` and `xend` shall also be null.

— If `xnext` is not a null pointer and `xnext < xend` for an output sequence, then a *write position* is available. In this case, `*xnext` shall be assignable as the next element to write (to put, or to store a character value, into the sequence).

— If `xnext` is not a null pointer and `xbeg < xnext` for an input sequence, then a *putback position* is available. In this case, `xnext[-1]` shall have a defined value and is the next (preceding) element to store a character that is put back into the input sequence.

— If `xnext` is not a null pointer and `xnext < xend` for an input sequence, then a *read position* is available. In this case, `*xnext` shall have a defined value and is the next element to read (to get, or to obtain a character value, from the sequence).

27.6.3 Class template `basic_streambuf<charT,traits>` [streambuf]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_streambuf {
  public:

    // types:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;

    virtual ~basic_streambuf();

    // 27.6.3.2.1 locales:
    locale   pubimbue(const locale& loc);
    locale   getloc() const;

    // 27.6.3.2.2 buffer and positioning:
```



```
  basic_streambuf<char_type,traits>*
     pubsetbuf(char_type* s, streamsize n);
  pos_type pubseekoff(off_type off, ios_base::seekdir way,
    ios_base::openmode which =
        ios_base::in | ios_base::out);
  pos_type pubseekpos(pos_type sp,
    ios_base::openmode which =
        ios_base::in | ios_base::out);
  int      pubsync();

  // Get and put areas:
  // 27.6.3.2.3 Get area:
  streamsize in_avail();
  int_type snextc();
  int_type sbumpc();
  int_type sgetc();
  streamsize sgetn(char_type* s, streamsize n);

  // 27.6.3.2.4 Putback:
  int_type sputbackc(char_type c);
  int_type sungetc();

  // 27.6.3.2.5 Put area:
  int_type   sputc(char_type c);
  streamsize sputn(const char_type* s, streamsize n);

protected:
  basic_streambuf();
  basic_streambuf(const basic_streambuf& rhs);
  basic_streambuf& operator=(const basic_streambuf& rhs);

  void swap(basic_streambuf& rhs);

  // 27.6.3.3.2 Get area:
  char_type* eback() const;
  char_type* gptr()  const;
  char_type* egptr() const;
  void       gbump(int n);
  void       setg(char_type* gbeg, char_type* gnext, char_type* gend);

  // 27.6.3.3.3 Put area:
  char_type* pbase() const;
  char_type* pptr() const;
  char_type* epptr() const;
  void       pbump(int n);
  void       setp(char_type* pbeg, char_type* pend);

  // 27.6.3.4 virtual functions:
  // 27.6.3.4.1 Locales:
  virtual void imbue(const locale& loc);

  // 27.6.3.4.2 Buffer management and positioning:
  virtual basic_streambuf<char_type,traits>*
       setbuf(char_type* s, streamsize n);
  virtual pos_type seekoff(off_type off, ios_base::seekdir way,
```



```
        ios_base::openmode which = ios_base::in | ios_base::out);
    virtual pos_type seekpos(pos_type sp,
        ios_base::openmode which = ios_base::in | ios_base::out);
    virtual int      sync();

    // 27.6.3.4.3 Get area:
    virtual streamsize showmanyc();
    virtual streamsize xsgetn(char_type* s, streamsize n);
    virtual int_type   underflow();
    virtual int_type   uflow();

    // 27.6.3.4.4 Putback:
    virtual int_type   pbackfail(int_type c = traits::eof());

    // 27.6.3.4.5 Put area:
    virtual streamsize xsputn(const char_type* s, streamsize n);
    virtual int_type   overflow (int_type c = traits::eof());
  };
}
```

1 The class template `basic_streambuf<charT,traits>` serves as an abstract base class for deriving various *stream buffers* whose objects each control two *character sequences*:

— a character *input sequence*;

— a character *output sequence*.

27.6.3.1 `basic_streambuf` constructors [streambuf.cons]

```
basic_streambuf();
```

1 *Effects:* Constructs an object of class `basic_streambuf<charT,traits>` and initializes:[305]

— all its pointer member objects to null pointers,

— the `getloc()` member to a copy the global locale, `locale()`, at the time of construction.

2 *Remarks:* Once the `getloc()` member is initialized, results of calling locale member functions, and of members of facets so obtained, can safely be cached until the next time the member `imbue` is called.

```
basic_streambuf(const basic_streambuf& rhs);
```

3 *Effects:* Constructs a copy of `rhs`.

4 *Postconditions:*

— `eback() == rhs.eback()`

— `gptr() == rhs.gptr()`

— `egptr() == rhs.egptr()`

— `pbase() == rhs.pbase()`

— `pptr() == rhs.pptr()`

— `epptr() == rhs.epptr()`

— `getloc() == rhs.getloc()`

305) The default constructor is protected for class `basic_streambuf` to assure that only objects for classes derived from this class may be constructed.


```
~basic_streambuf();
```

5 *Effects:* None.

27.6.3.2 `basic_streambuf` public member functions [streambuf.members]

27.6.3.2.1 Locales [streambuf.locales]

```
locale pubimbue(const locale& loc);
```

1 *Postcondition:* `loc == getloc()`.

2 *Effects:* Calls `imbue(loc)`.

3 *Returns:* Previous value of `getloc()`.

```
locale getloc() const;
```

4 *Returns:* If `pubimbue()` has ever been called, then the last value of `loc` supplied, otherwise the current global locale, `locale()`, in effect at the time of construction. If called after `pubimbue()` has been called but before `pubimbue` has returned (i.e., from within the call of `imbue()`) then it returns the previous value.

27.6.3.2.2 Buffer management and positioning [streambuf.buffer]

```
basic_streambuf<char_type,traits>* pubsetbuf(char_type* s, streamsize n);
```

1 *Returns:* `setbuf(s, n)`.

```
pos_type pubseekoff(off_type off, ios_base::seekdir way,
                ios_base::openmode which = ios_base::in | ios_base::out);
```

2 *Returns:* `seekoff(off, way, which)`.

```
pos_type pubseekpos(pos_type sp,
                ios_base::openmode which = ios_base::in | ios_base::out);
```

3 *Returns:* `seekpos(sp, which)`.

```
int pubsync();
```

4 *Returns:* `sync()`.

27.6.3.2.3 Get area [streambuf.pub.get]

```
streamsize in_avail();
```

1 *Returns:* If a read position is available, returns `egptr() - gptr()`. Otherwise returns `showmanyc()` (27.6.3.4.3).

```
int_type snextc();
```

2 *Effects:* Calls `sbumpc()`.

3 *Returns:* If that function returns `traits::eof()`, returns `traits::eof()`. Otherwise, returns `sgetc()`.


```
int_type sbumpc();
```

4 *Returns:* If the input sequence read position is not available, returns `uflow()`. Otherwise, returns `traits::to_int_type(*gptr())` and increments the next pointer for the input sequence.

```
int_type sgetc();
```

5 *Returns:* If the input sequence read position is not available, returns `underflow()`. Otherwise, returns `traits::to_int_type(*gptr())`.

```
streamsize sgetn(char_type* s, streamsize n);
```

6 *Returns:* `xsgetn(s, n)`.

27.6.3.2.4 Putback [streambuf.pub.pback]

```
int_type sputbackc(char_type c);
```

1 *Returns:* If the input sequence putback position is not available, or if `traits::eq(c,gptr()[-1])` is false, returns `pbackfail(traits::to_int_type(c))`. Otherwise, decrements the next pointer for the input sequence and returns `traits::to_int_type(*gptr())`.

```
int_type sungetc();
```

2 *Returns:* If the input sequence putback position is not available, returns `pbackfail()`. Otherwise, decrements the next pointer for the input sequence and returns `traits::to_int_type(*gptr())`.

27.6.3.2.5 Put area [streambuf.pub.put]

```
int_type sputc(char_type c);
```

1 *Returns:* If the output sequence write position is not available, returns `overflow(traits::to_int_type(c))`. Otherwise, stores `c` at the next pointer for the output sequence, increments the pointer, and returns `traits::to_int_type(c)`.

```
streamsize sputn(const char_type* s, streamsize n);
```

2 *Returns:* `xsputn(s,n)`.

27.6.3.3 `basic_streambuf` protected member functions [streambuf.protected]

27.6.3.3.1 Assignment [streambuf.assign]

```
basic_streambuf& operator=(const basic_streambuf& rhs);
```

1 *Effects:* Assigns the data members of `rhs` to `*this`.

2 *Postconditions:*

- `eback() == rhs.eback()`
- `gptr() == rhs.gptr()`
- `egptr() == rhs.egptr()`
- `pbase() == rhs.pbase()`
- `pptr() == rhs.pptr()`

— `epptr() == rhs.epptr()`
— `getloc() == rhs.getloc()`

3 *Returns:* `*this`.

```
void swap(basic_streambuf& rhs);
```

4 *Effects:* Swaps the data members of `rhs` and `*this`.

27.6.3.3.2 Get area access [streambuf.get.area]

```
char_type* eback() const;
```

1 *Returns:* The beginning pointer for the input sequence.

```
char_type* gptr() const;
```

2 *Returns:* The next pointer for the input sequence.

```
char_type* egptr() const;
```

3 *Returns:* The end pointer for the input sequence.

```
void gbump(int n);
```

4 *Effects:* Adds `n` to the next pointer for the input sequence.

```
void setg(char_type* gbeg, char_type* gnext, char_type* gend);
```

5 *Postconditions:* `gbeg == eback()`, `gnext == gptr()`, and `gend == egptr()`.

27.6.3.3.3 Put area access [streambuf.put.area]

```
char_type* pbase() const;
```

1 *Returns:* The beginning pointer for the output sequence.

```
char_type* pptr() const;
```

2 *Returns:* The next pointer for the output sequence.

```
char_type* epptr() const;
```

3 *Returns:* The end pointer for the output sequence.

```
void pbump(int n);
```

4 *Effects:* Adds `n` to the next pointer for the output sequence.

```
void setp(char_type* pbeg, char_type* pend);
```

5 *Postconditions:* `pbeg == pbase()`, `pbeg == pptr()`, and `pend == epptr()`.

27.6.3.4 `basic_streambuf` virtual functions [streambuf.virtuals]

27.6.3.4.1 Locales [streambuf.virt.locales]

```
void imbue(const locale&)
```

1 *Effects:* Change any translations based on locale.

2 *Remarks:* Allows the derived class to be informed of changes in locale at the time they occur. Between invocations of this function a class derived from streambuf can safely cache results of calls to locale functions and to members of facets so obtained.

3 *Default behavior:* Does nothing.

27.6.3.4.2 Buffer management and positioning [streambuf.virt.buffer]

```
basic_streambuf* setbuf(char_type* s, streamsize n);
```

1 *Effects:* Influences stream buffering in a way that is defined separately for each class derived from `basic_streambuf` in this Clause (27.8.2.4, 27.9.1.5).

2 *Default behavior:* Does nothing. Returns `this`.

```
pos_type seekoff(off_type off, ios_base::seekdir way,
                 ios_base::openmode which
                  = ios_base::in | ios_base::out);
```

3 *Effects:* Alters the stream positions within one or more of the controlled sequences in a way that is defined separately for each class derived from `basic_streambuf` in this Clause (27.8.2.4, 27.9.1.5).

4 *Default behavior:* Returns `pos_type(off_type(-1))`.

```
pos_type seekpos(pos_type sp,
                 ios_base::openmode which
                  = ios_base::in | ios_base::out);
```

5 *Effects:* Alters the stream positions within one or more of the controlled sequences in a way that is defined separately for each class derived from `basic_streambuf` in this Clause (27.8.2, 27.9.1.1).

6 *Default behavior:* Returns `pos_type(off_type(-1))`.

```
int sync();
```

7 *Effects:* Synchronizes the controlled sequences with the arrays. That is, if `pbase()` is non-null the characters between `pbase()` and `pptr()` are written to the controlled sequence. The pointers may then be reset as appropriate.

8 *Returns:* -1 on failure. What constitutes failure is determined by each derived class (27.9.1.5).

9 *Default behavior:* Returns zero.

27.6.3.4.3 Get area [streambuf.virt.get]

```
streamsize showmanyc();[306]
```

306) The morphemes of `showmanyc` are "es-how-many-see", not "show-manic".

1 *Returns:* An estimate of the number of characters available in the sequence, or -1. If it returns a positive value, then successive calls to `underflow()` will not return `traits::eof()` until at least that number of characters have been extracted from the stream. If `showmanyc()` returns -1, then calls to `underflow()` or `uflow()` will fail.[307]

2 *Default behavior:* Returns zero.

3 *Remarks:* Uses `traits::eof()`.

```
streamsize xsgetn(char_type* s, streamsize n);
```

4 *Effects:* Assigns up to `n` characters to successive elements of the array whose first element is designated by `s`. The characters assigned are read from the input sequence as if by repeated calls to `sbumpc()`. Assigning stops when either `n` characters have been assigned or a call to `sbumpc()` would return `traits::eof()`.

5 *Returns:* The number of characters assigned.[308]

6 *Remarks:* Uses `traits::eof()`.

```
int_type underflow();
```

7 *Remarks:* The public members of `basic_streambuf` call this virtual function only if `gptr()` is null or `gptr() >= egptr()`

8 *Returns:* `traits::to_int_type(c)`, where `c` is the first *character* of the *pending sequence*, without moving the input sequence position past it. If the pending sequence is null then the function returns `traits::eof()` to indicate failure.

9 The *pending sequence* of characters is defined as the concatenation of:

a) If `gptr()` is non-null, then the `egptr() - gptr()` characters starting at `gptr()`, otherwise the empty sequence.

b) Some sequence (possibly empty) of characters read from the input sequence.

10 The *result character* is

a) If the pending sequence is non-empty, the first character of the sequence.

b) If the pending sequence is empty then the next character that would be read from the input sequence.

11 The *backup sequence* is defined as the concatenation of:

a) If `eback()` is null then empty,

b) Otherwise the `gptr() - eback()` characters beginning at `eback()`.

12 *Effects:* The function sets up the `gptr()` and `egptr()` satisfying one of:

a) If the pending sequence is non-empty, `egptr()` is non-null and `egptr() - gptr()` characters starting at `gptr()` are the characters in the pending sequence

b) If the pending sequence is empty, either `gptr()` is null or `gptr()` and `egptr()` are set to the same non-null pointer value.

307) `underflow` or `uflow` might fail by throwing an exception prematurely. The intention is not only that the calls will not return `eof()` but that they will return "immediately."

308) Classes derived from `basic_streambuf` can provide more efficient ways to implement `xsgetn()` and `xsputn()` by overriding these definitions from the base class.

13 If `eback()` and `gptr()` are non-null then the function is not constrained as to their contents, but the "usual backup condition" is that either:

a) If the backup sequence contains at least `gptr() - eback()` characters, then the `gptr() - eback()` characters starting at `eback()` agree with the last `gptr() - eback()` characters of the backup sequence.

b) Or the `n` characters starting at `gptr() - n` agree with the backup sequence (where `n` is the length of the backup sequence)

14 *Default behavior:* Returns `traits::eof()`.

```
int_type uflow();
```

15 *Requires:* The constraints are the same as for `underflow()`, except that the result character shall be transferred from the pending sequence to the backup sequence, and the pending sequence shall not be empty before the transfer.

16 *Default behavior:* Calls `underflow()`. If `underflow()` returns `traits::eof()`, returns `traits::eof()`. Otherwise, returns the value of `traits::to_int_type(*gptr())` and increment the value of the next pointer for the input sequence.

17 *Returns:* `traits::eof()` to indicate failure.

27.6.3.4.4 Putback [streambuf.virt.pback]

```
int_type pbackfail(int_type c = traits::eof());
```

1 *Remarks:* The public functions of `basic_streambuf` call this virtual function only when `gptr()` is null, `gptr() == eback()`, or `traits::eq(traits::to_char_type(c),gptr()[-1])` returns `false`. Other calls shall also satisfy that constraint.

The *pending sequence* is defined as for `underflow()`, with the modifications that

— If `traits::eq_int_type(c,traits::eof())` returns `true`, then the input sequence is backed up one character before the pending sequence is determined.

— If `traits::eq_int_type(c,traits::eof())` returns `false`, then `c` is prepended. Whether the input sequence is backed up or modified in any other way is unspecified.

2 *Postcondition:* On return, the constraints of `gptr()`, `eback()`, and `pptr()` are the same as for `underflow()`.

3 *Returns:* `traits::eof()` to indicate failure. Failure may occur because the input sequence could not be backed up, or if for some other reason the pointers could not be set consistent with the constraints. `pbackfail()` is called only when put back has really failed.

4 Returns some value other than `traits::eof()` to indicate success.

5 *Default behavior:* Returns `traits::eof()`.

27.6.3.4.5 Put area [streambuf.virt.put]

```
streamsize xsputn(const char_type* s, streamsize n);
```

1 *Effects:* Writes up to `n` characters to the output sequence as if by repeated calls to `sputc(c)`. The characters written are obtained from successive elements of the array whose first element is designated by `s`. Writing stops when either `n` characters have been written or a call to `sputc(c)` would return `traits::eof()`. Is is unspecified whether the function calls `overflow()` when `pptr() == epptr()` becomes true or whether it achieves the same effects by other means.

2 *Returns:* The number of characters written.


```
int_type overflow(int_type c = traits::eof());
```

3 *Effects:* Consumes some initial subsequence of the characters of the *pending sequence*. The pending sequence is defined as the concatenation of

a) if `pbase()` is null then the empty sequence otherwise, `pptr() - pbase()` characters beginning at `pbase()`.

b) if `traits::eq_int_type(c,traits::eof())` returns `true`, then the empty sequence otherwise, the sequence consisting of `c`.

4 *Remarks:* The member functions `sputc()` and `sputn()` call this function in case that no room can be found in the put buffer enough to accommodate the argument character sequence.

5 *Requires:* Every overriding definition of this virtual function shall obey the following constraints:

1) The effect of consuming a character on the associated output sequence is specified[309]

2) Let `r` be the number of characters in the pending sequence not consumed. If `r` is non-zero then `pbase()` and `pptr()` shall be set so that: `pptr() - pbase() == r` and the `r` characters starting at `pbase()` are the associated output stream. In case `r` is zero (all characters of the pending sequence have been consumed) then either `pbase()` is set to `nullptr`, or `pbase()` and `pptr()` are both set to the same non-null value.

3) The function may fail if either appending some character to the associated output stream fails or if it is unable to establish `pbase()` and `pptr()` according to the above rules.

6 *Returns:* `traits::eof()` or throws an exception if the function fails.

Otherwise, returns some value other than `traits::eof()` to indicate success.[310]

7 *Default behavior:* Returns `traits::eof()`.

27.7 Formatting and manipulators [iostream.format]

27.7.1 Overview [iostream.format.overview]

Header `<istream>` synopsis

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
    class basic_istream;
  typedef basic_istream<char>     istream;
  typedef basic_istream<wchar_t> wistream;

  template <class charT, class traits = char_traits<charT> >
    class basic_iostream;
  typedef basic_iostream<char>    iostream;
  typedef basic_iostream<wchar_t> wiostream;

  template <class charT, class traits>
    basic_istream<charT,traits>& ws(basic_istream<charT,traits>& is);

  template <class charT, class traits, class T>
    basic_istream<charT, traits>&
    operator>>(basic_istream<charT, traits>&& is, T& x);
}
```

309) That is, for each class derived from an instance of `basic_streambuf` in this Clause (27.8.2, 27.9.1.1), a specification of how consuming a character effects the associated output sequence is given. There is no requirement on a program-defined class.

310) Typically, `overflow` returns `c` to indicate success, except when `traits::eq_int_type(c,traits::eof())` returns `true`, in which case it returns `traits::not_eof(c)`.

Header <ostream> synopsis

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
    class basic_ostream;
  typedef basic_ostream<char>     ostream;
  typedef basic_ostream<wchar_t> wostream;

  template <class charT, class traits>
    basic_ostream<charT,traits>& endl(basic_ostream<charT,traits>& os);
  template <class charT, class traits>
    basic_ostream<charT,traits>& ends(basic_ostream<charT,traits>& os);
  template <class charT, class traits>
    basic_ostream<charT,traits>& flush(basic_ostream<charT,traits>& os);

  template <class charT, class traits, class T>
    basic_ostream<charT, traits>&
    operator<<(basic_ostream<charT, traits>&& os, const T& x);
}
```

Header <iomanip> synopsis

```
namespace std {
  // types T1, T2, ... are unspecified implementation types
  T1 resetiosflags(ios_base::fmtflags mask);
  T2 setiosflags  (ios_base::fmtflags mask);
  T3 setbase(int base);
  template<charT> T4 setfill(charT c);
  T5 setprecision(int n);
  T6 setw(int n);
  template <class moneyT> T7 get_money(moneyT& mon, bool intl = false);
  template <class moneyT> T8 put_money(const moneyT& mon, bool intl = false);
  template <class charT> T9 get_time(struct tm* tmb, const charT* fmt);
  template <class charT> T10 put_time(const struct tm* tmb, const charT* fmt);

  template <class charT>
    T11 quoted(const charT* s, charT delim=charT('"'), charT escape=charT('\\'));

  template <class charT, class traits, class Allocator>
    T12 quoted(const basic_string<charT, traits, Allocator>& s,
               charT delim=charT('"'), charT escape=charT('\\'));

  template <class charT, class traits, class Allocator>
    T13 quoted(basic_string<charT, traits, Allocator>& s,
               charT delim=charT('"'), charT escape=charT('\\'));
}
```

27.7.2 Input streams [input.streams]

1 The header <istream> defines two types and a function signature that control input from a stream buffer along with a function template that extracts from stream rvalues.

27.7.2.1 Class template basic_istream [istream]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_istream : virtual public basic_ios<charT,traits> {
  public:
    // types (inherited from basic_ios (27.5.5)):
```



```
typedef charT                     char_type;
typedef typename traits::int_type int_type;
typedef typename traits::pos_type pos_type;
typedef typename traits::off_type off_type;
typedef traits                    traits_type;

// 27.7.2.1.1 Constructor/destructor:
explicit basic_istream(basic_streambuf<charT,traits>* sb);
virtual ~basic_istream();

// 27.7.2.1.3 Prefix/suffix:
class sentry;

// 27.7.2.2 Formatted input:
basic_istream<charT,traits>& operator>>(
  basic_istream<charT,traits>& (*pf)(basic_istream<charT,traits>&));
basic_istream<charT,traits>& operator>>(
                    basic_ios<charT,traits>& (*pf)(basic_ios<charT,traits>&));
basic_istream<charT,traits>& operator>>(
  ios_base& (*pf)(ios_base&));

basic_istream<charT,traits>& operator>>(bool& n);
basic_istream<charT,traits>& operator>>(short& n);
basic_istream<charT,traits>& operator>>(unsigned short& n);
basic_istream<charT,traits>& operator>>(int& n);
basic_istream<charT,traits>& operator>>(unsigned int& n);
basic_istream<charT,traits>& operator>>(long& n);
basic_istream<charT,traits>& operator>>(unsigned long& n);
basic_istream<charT,traits>& operator>>(long long& n);
basic_istream<charT,traits>& operator>>(unsigned long long& n);
basic_istream<charT,traits>& operator>>(float& f);
basic_istream<charT,traits>& operator>>(double& f);
basic_istream<charT,traits>& operator>>(long double& f);

basic_istream<charT,traits>& operator>>(void*& p);
basic_istream<charT,traits>& operator>>(
  basic_streambuf<char_type,traits>* sb);

// 27.7.2.3 Unformatted input:
streamsize gcount() const;
int_type get();
basic_istream<charT,traits>& get(char_type& c);
basic_istream<charT,traits>& get(char_type* s, streamsize n);
basic_istream<charT,traits>& get(char_type* s, streamsize n,
                                 char_type delim);
basic_istream<charT,traits>& get(basic_streambuf<char_type,traits>& sb);
basic_istream<charT,traits>& get(basic_streambuf<char_type,traits>& sb,
                                 char_type delim);

basic_istream<charT,traits>& getline(char_type* s, streamsize n);
basic_istream<charT,traits>& getline(char_type* s, streamsize n,
                                     char_type delim);

basic_istream<charT,traits>& ignore(
  streamsize n = 1, int_type delim = traits::eof());
```



```
    int_type                    peek();
    basic_istream<charT,traits>& read    (char_type* s, streamsize n);
    streamsize                  readsome(char_type* s, streamsize n);

    basic_istream<charT,traits>& putback(char_type c);
    basic_istream<charT,traits>& unget();
    int sync();

    pos_type tellg();
    basic_istream<charT,traits>& seekg(pos_type);
    basic_istream<charT,traits>& seekg(off_type, ios_base::seekdir);

  protected:
    basic_istream(const basic_istream& rhs) = delete;
    basic_istream(basic_istream&& rhs);

    // 27.7.2.1.2 Assign/swap:
    basic_istream& operator=(const basic_istream& rhs) = delete;
    basic_istream& operator=(basic_istream&& rhs);
    void swap(basic_istream& rhs);
  };

  // 27.7.2.2.3 character extraction templates:
  template<class charT, class traits>
    basic_istream<charT,traits>& operator>>(basic_istream<charT,traits>&,
                                            charT&);
  template<class traits>
    basic_istream<char,traits>& operator>>(basic_istream<char,traits>&,
                                           unsigned char&);
  template<class traits>
    basic_istream<char,traits>& operator>>(basic_istream<char,traits>&,
                                           signed char&);

  template<class charT, class traits>
    basic_istream<charT,traits>& operator>>(basic_istream<charT,traits>&,
                                            charT*);
  template<class traits>
    basic_istream<char,traits>& operator>>(basic_istream<char,traits>&,
                                           unsigned char*);
  template<class traits>
    basic_istream<char,traits>& operator>>(basic_istream<char,traits>&,
                                           signed char*);
}
```

1 The class `basic_istream` defines a number of member function signatures that assist in reading and interpreting input from sequences controlled by a stream buffer.

2 Two groups of member function signatures share common properties: the *formatted input functions* (or *extractors*) and the *unformatted input functions*. Both groups of input functions are described as if they obtain (or *extract*) input *characters* by calling `rdbuf()->sbumpc()` or `rdbuf()->sgetc()`. They may use other public members of `istream`.

3 If `rdbuf()->sbumpc()` or `rdbuf()->sgetc()` returns `traits::eof()`, then the input function, except as explicitly noted otherwise, completes its actions and does `setstate(eofbit)`, which may throw `ios_base::failure` (27.5.5.4), before returning.

4 If one of these called functions throws an exception, then unless explicitly noted otherwise, the input function sets `badbit` in error state. If `badbit` is on in `exceptions()`, the input function rethrows the exception

without completing its actions, otherwise it does not throw anything and proceeds as if the called function had returned a failure indication.

27.7.2.1.1 `basic_istream` constructors [istream.cons]

```
explicit basic_istream(basic_streambuf<charT,traits>* sb);
```

1 *Effects:* Constructs an object of class `basic_istream`, assigning initial values to the base class by calling `basic_ios::init(sb)` (27.5.5.2).

2 *Postcondition:* `gcount() == 0`

```
basic_istream(basic_istream&& rhs);
```

3 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by default constructing the base class, copying the `gcount()` from `rhs`, calling `basic_ios<charT, traits>::move(rhs)` to initialize the base class, and setting the `gcount()` for `rhs` to 0.

```
virtual ~basic_istream();
```

4 *Effects:* Destroys an object of class `basic_istream`.

5 *Remarks:* Does not perform any operations of `rdbuf()`.

27.7.2.1.2 Class `basic_istream` assign and swap [istream.assign]

```
basic_istream& operator=(basic_istream&& rhs);
```

1 *Effects:* `swap(rhs);`.

2 *Returns:* `*this`.

```
void swap(basic_istream& rhs);
```

3 *Effects:* Calls `basic_ios<charT, traits>::swap(rhs)`. Exchanges the values returned by `gcount()` and `rhs.gcount()`.

27.7.2.1.3 Class `basic_istream::sentry` [istream::sentry]

```
namespace std {
  template <class charT,class traits = char_traits<charT> >
  class basic_istream<charT,traits>::sentry {
    typedef traits traits_type;
    bool ok_; // exposition only
  public:
    explicit sentry(basic_istream<charT,traits>& is, bool noskipws = false);
    ~sentry();
    explicit operator bool() const { return ok_; }
    sentry(const sentry&) = delete;
    sentry& operator=(const sentry&) = delete;
  };
}
```

1 The class `sentry` defines a class that is responsible for doing exception safe prefix and suffix operations.

```
explicit sentry(basic_istream<charT,traits>& is, bool noskipws = false);
```


2 *Effects:* If `is.good()` is `false`, calls `is.setstate(failbit)`. Otherwise, prepares for formatted or unformatted input. First, if `is.tie()` is not a null pointer, the function calls `is.tie()->flush()` to synchronize the output sequence with any associated external C stream. Except that this call can be suppressed if the put area of `is.tie()` is empty. Further an implementation is allowed to defer the call to `flush` until a call of `is.rdbuf()->underflow()` occurs. If no such call occurs before the `sentry` object is destroyed, the call to `flush` may be eliminated entirely.[311] If `noskipws` is zero and `is.flags() & ios_base::skipws` is nonzero, the function extracts and discards each character as long as the next available input character `c` is a whitespace character. If `is.rdbuf()->sbumpc()` or `is.rdbuf()->sgetc()` returns `traits::eof()`, the function calls `setstate(failbit | eofbit)` (which may throw `ios_base::failure`).

3 *Remarks:* The constructor `explicit sentry(basic_istream<charT,traits>& is, bool noskipws = false)` uses the currently imbued locale in `is`, to determine whether the next input character is whitespace or not.

4 To decide if the character `c` is a whitespace character, the constructor performs as if it executes the following code fragment:

```
const ctype<charT>& ctype = use_facet<ctype<charT> >(is.getloc());
if (ctype.is(ctype.space,c)!=0)
  // c is a whitespace character.
```

5 If, after any preparation is completed, `is.good()` is `true`, `ok_ != false` otherwise, `ok_ == false`. During preparation, the constructor may call `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4))[312]

```
~sentry();
```

6 *Effects:* None.

```
explicit operator bool() const;
```

7 *Effects:* Returns `ok_`.

27.7.2.2 Formatted input functions [istream.formatted]

27.7.2.2.1 Common requirements [istream.formatted.reqmts]

1 Each formatted input function begins execution by constructing an object of class `sentry` with the `noskipws` (second) argument `false`. If the `sentry` object returns `true`, when converted to a value of type `bool`, the function endeavors to obtain the requested input. If an exception is thrown during input then `ios::badbit` is turned on[313] in `*this`'s error state. If `(exceptions()&badbit) != 0` then the exception is rethrown. In any case, the formatted input function destroys the `sentry` object. If no exception has been thrown, it returns `*this`.

27.7.2.2.2 Arithmetic extractors [istream.formatted.arithmetic]

```
operator>>(unsigned short& val);
operator>>(unsigned int& val);
operator>>(long& val);
operator>>(unsigned long& val);
```

311) This will be possible only in functions that are part of the library. The semantics of the constructor used in user code is as specified.

312) The sentry constructor and destructor can also perform additional implementation-dependent operations.

313) This is done without causing an `ios::failure` to be thrown.


```
operator>>(long long& val);
operator>>(unsigned long long& val);
operator>>(float& val);
operator>>(double& val);
operator>>(long double& val);
operator>>(bool& val);
operator>>(void*& val);
```

1 As in the case of the inserters, these extractors depend on the locale's `num_get<>` (22.4.2.1) object to perform parsing the input stream data. These extractors behave as formatted input functions (as described in 27.7.2.2.1). After a sentry object is constructed, the conversion occurs as if performed by the following code fragment:

```
typedef num_get< charT,istreambuf_iterator<charT,traits> > numget;
iostate err = iostate::goodbit;
use_facet< numget >(loc).get(*this, 0, *this, err, val);
setstate(err);
```

In the above fragment, `loc` stands for the private member of the `basic_ios` class. [*Note:* The first argument provides an object of the `istreambuf_iterator` class which is an iterator pointed to an input stream. It bypasses istreams and uses streambufs directly. — *end note*] Class `locale` relies on this type as its interface to `istream`, so that it does not need to depend directly on `istream`.

```
operator>>(short& val);
```

2 The conversion occurs as if performed by the following code fragment (using the same notation as for the preceding code fragment):

```
typedef num_get<charT,istreambuf_iterator<charT,traits> > numget;
iostate err = ios_base::goodbit;
long lval;
use_facet<numget>(loc).get(*this, 0, *this, err, lval);
if (lval < numeric_limits<short>::min()) {
  err |= ios_base::failbit;
  val = numeric_limits<short>::min();
} else if (numeric_limits<short>::max() < lval) {
  err |= ios_base::failbit;
  val = numeric_limits<short>::max();
}  else
  val = static_cast<short>(lval);
setstate(err);
```

```
operator>>(int& val);
```

3 The conversion occurs as if performed by the following code fragment (using the same notation as for the preceding code fragment):

```
typedef num_get<charT,istreambuf_iterator<charT,traits> > numget;
iostate err = ios_base::goodbit;
long lval;
use_facet<numget>(loc).get(*this, 0, *this, err, lval);
if (lval < numeric_limits<int>::min()) {
  err |= ios_base::failbit;
  val = numeric_limits<int>::min();
} else if (numeric_limits<int>::max() < lval) {
```



```
      err |= ios_base::failbit;
      val = numeric_limits<int>::max();
    }  else
      val = static_cast<int>(lval);
    setstate(err);
```

27.7.2.2.3 `basic_istream::operator>>` [istream::extractors]

```
basic_istream<charT,traits>& operator>>
    (basic_istream<charT,traits>& (*pf)(basic_istream<charT,traits>&))
```

1 *Effects:* None. This extractor does not behave as a formatted input function (as described in 27.7.2.2.1.)

2 *Returns:* `pf(*this)`.[314]

```
basic_istream<charT,traits>& operator>>
    (basic_ios<charT,traits>& (*pf)(basic_ios<charT,traits>&));
```

3 *Effects:* Calls `pf(*this)`. This extractor does not behave as a formatted input function (as described in 27.7.2.2.1).

4 *Returns:* `*this`.

```
basic_istream<charT,traits>& operator>>
    (ios_base& (*pf)(ios_base&));
```

5 *Effects:* Calls `pf(*this)`.[315] This extractor does not behave as a formatted input function (as described in 27.7.2.2.1).

6 *Returns:* `*this`.

```
template<class charT, class traits>
  basic_istream<charT,traits>& operator>>(basic_istream<charT,traits>& in,
                                          charT* s);
template<class traits>
  basic_istream<char,traits>& operator>>(basic_istream<char,traits>& in,
                                         unsigned char* s);
template<class traits>
  basic_istream<char,traits>& operator>>(basic_istream<char,traits>& in,
                                         signed char* s);
```

7 *Effects:* Behaves like a formatted input member (as described in 27.7.2.2.1) of `in`. After a `sentry` object is constructed, `operator>>` extracts characters and stores them into successive locations of an array whose first element is designated by `s`. If `width()` is greater than zero, `n` is `width()`. Otherwise `n` is the number of elements of the largest array of `char_type` that can store a terminating `charT()`. `n` is the maximum number of characters stored.

8 Characters are extracted and stored until any of the following occurs:

— `n-1` characters are stored;

— end of file occurs on the input sequence;

314) See, for example, the function signature `ws(basic_istream&)` (27.7.2.4).
315) See, for example, the function signature `dec(ios_base&)` (27.5.6.3).

— `ct.is(ct.space,c)` is `true` for the next available input character `c`, where `ct` is `use_facet<ctype< charT> >(in.getloc())`.

9 `operator>>` then stores a null byte (`charT()`) in the next position, which may be the first position if no characters were extracted. `operator>>` then calls `width(0)`.

10 If the function extracted no characters, it calls `setstate(failbit)`, which may throw `ios_base::failure` (27.5.5.4).

11 *Returns:* `in`.

```
template<class charT, class traits>
  basic_istream<charT,traits>& operator>>(basic_istream<charT,traits>& in,
                                          charT& c);
template<class traits>
  basic_istream<char,traits>& operator>>(basic_istream<char,traits>& in,
                                         unsigned char& c);
template<class traits>
  basic_istream<char,traits>& operator>>(basic_istream<char,traits>& in,
                                         signed char& c);
```

12 *Effects:* Behaves like a formatted input member (as described in 27.7.2.2.1) of `in`. After a `sentry` object is constructed a character is extracted from `in`, if one is available, and stored in `c`. Otherwise, the function calls `in.setstate(failbit)`.

13 *Returns:* `in`.

```
basic_istream<charT,traits>& operator>>
  (basic_streambuf<charT,traits>* sb);
```

14 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). If `sb` is null, calls `setstate(failbit)`, which may throw `ios_base::failure` (27.5.5.4). After a sentry object is constructed, extracts characters from `*this` and inserts them in the output sequence controlled by `sb`. Characters are extracted and inserted until any of the following occurs:

— end-of-file occurs on the input sequence;

— inserting in the output sequence fails (in which case the character to be inserted is not extracted);

— an exception occurs (in which case the exception is caught).

15 If the function inserts no characters, it calls `setstate(failbit)`, which may throw `ios_base::failure` (27.5.5.4). If it inserted no characters because it caught an exception thrown while extracting characters from `*this` and `failbit` is on in `exceptions()` (27.5.5.4), then the caught exception is rethrown.

16 *Returns:* `*this`.

27.7.2.3 Unformatted input functions [istream.unformatted]

1 Each unformatted input function begins execution by constructing an object of class `sentry` with the default argument `noskipws` (second) argument `true`. If the `sentry` object returns `true`, when converted to a value of type `bool`, the function endeavors to obtain the requested input. Otherwise, if the sentry constructor exits by throwing an exception or if the sentry object returns false, when converted to a value of type `bool`, the function returns without attempting to obtain any input. In either case the number of extracted characters is set to 0; unformatted input functions taking a character array of non-zero size as an argument shall also store a null character (using `charT()`) in the first location of the array. If an exception is thrown during input then `ios::badbit` is turned on[316] in `*this`'s error state. (Exceptions thrown from `basic_ios<>::clear()`

316) This is done without causing an `ios::failure` to be thrown.

are not caught or rethrown.) If `(exceptions()&badbit) != 0` then the exception is rethrown. It also counts the number of characters extracted. If no exception has been thrown it ends by storing the count in a member object and returning the value specified. In any event the `sentry` object is destroyed before leaving the unformatted input function.

```
streamsize gcount() const;
```

2 *Effects:* None. This member function does not behave as an unformatted input function (as described in 27.7.2.3, paragraph 1).

3 *Returns:* The number of characters extracted by the last unformatted input member function called for the object.

```
int_type get();
```

4 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, extracts a character `c`, if one is available. Otherwise, the function calls `setstate(failbit)`, which may throw `ios_base::failure` (27.5.5.4),

5 *Returns:* `c` if available, otherwise `traits::eof()`.

```
basic_istream<charT,traits>& get(char_type& c);
```

6 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, extracts a character, if one is available, and assigns it to `c`.[317] Otherwise, the function calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)).

7 *Returns:* `*this`.

```
basic_istream<charT,traits>& get(char_type* s, streamsize n,
                  char_type delim );
```

8 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, extracts characters and stores them into successive locations of an array whose first element is designated by `s`.[318] Characters are extracted and stored until any of the following occurs:

— `n` is less than one or `n - 1` characters are stored;

— end-of-file occurs on the input sequence (in which case the function calls `setstate(eofbit)`);

— `traits::eq(c, delim)` for the next available input character `c` (in which case `c` is not extracted).

9 If the function stores no characters, it calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)). In any case, if `n` is greater than zero it then stores a null character into the next successive location of the array.

10 *Returns:* `*this`.

```
basic_istream<charT,traits>& get(char_type* s, streamsize n)
```

11 *Effects:* Calls `get(s,n,widen('\n'))`

12 *Returns:* Value returned by the call.

317) Note that this function is not overloaded on types `signed char` and `unsigned char`.
318) Note that this function is not overloaded on types `signed char` and `unsigned char`.


```
basic_istream<charT,traits>& get(basic_streambuf<char_type,traits>& sb,
                  char_type delim );
```

13 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, extracts characters and inserts them in the output sequence controlled by `sb`. Characters are extracted and inserted until any of the following occurs:

- end-of-file occurs on the input sequence;
- inserting in the output sequence fails (in which case the character to be inserted is not extracted);
- `traits::eq(c, delim)` for the next available input character `c` (in which case `c` is not extracted);
- an exception occurs (in which case, the exception is caught but not rethrown).

14 If the function inserts no characters, it calls `setstate(failbit)`, which may throw `ios_base::failure` (27.5.5.4).

15 *Returns:* `*this`.

```
basic_istream<charT,traits>& get(basic_streambuf<char_type,traits>& sb);
```

16 *Effects:* Calls `get(sb, widen('\n'))`

17 *Returns:* Value returned by the call.

```
basic_istream<charT,traits>& getline(char_type* s, streamsize n,
                    char_type delim);
```

18 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, extracts characters and stores them into successive locations of an array whose first element is designated by `s`.[319] Characters are extracted and stored until one of the following occurs:

1. end-of-file occurs on the input sequence (in which case the function calls `setstate(eofbit)`);
2. `traits::eq(c, delim)` for the next available input character `c` (in which case the input character is extracted but not stored);[320]
3. `n` is less than one or `n - 1` characters are stored (in which case the function calls `setstate(failbit)`).

19 These conditions are tested in the order shown.[321]

20 If the function extracts no characters, it calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)).[322]

21 In any case, if `n` is greater than zero, it then stores a null character (using `charT()`) into the next successive location of the array.

22 *Returns:* `*this`.

23 [*Example:*

319) Note that this function is not overloaded on types `signed char` and `unsigned char`.
320) Since the final input character is "extracted," it is counted in the `gcount()`, even though it is not stored.
321) This allows an input line which exactly fills the buffer, without setting `failbit`. This is different behavior than the historical AT&T implementation.
322) This implies an empty input line will not cause `failbit` to be set.


```
#include <iostream>

int main() {
  using namespace std;
  const int line_buffer_size = 100;

  char buffer[line_buffer_size];
  int line_number = 0;
  while (cin.getline(buffer, line_buffer_size, '\n') || cin.gcount()) {
    int count = cin.gcount();
    if (cin.eof())
      cout << "Partial final line";    // cin.fail() is false
    else if (cin.fail()) {
      cout << "Partial long line";
      cin.clear(cin.rdstate() & ~ios_base::failbit);
    } else {
      count--;                          // Don't include newline in count
      cout << "Line " << ++line_number;
    }
    cout << " (" << count << " chars): " << buffer << endl;
  }
}
```

— *end example*]

```
basic_istream<charT,traits>& getline(char_type* s, streamsize n);
```

24 *Returns:* `getline(s,n,widen('\n'))`

```
basic_istream<charT,traits>&
    ignore(streamsize n = 1, int_type delim = traits::eof());
```

25 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, extracts characters and discards them. Characters are extracted until any of the following occurs:

- `n != numeric_limits<streamsize>::max()` (18.3.2) and `n` characters have been extracted so far
- end-of-file occurs on the input sequence (in which case the function calls `setstate(eofbit)`, which may throw `ios_base::failure` (27.5.5.4));
- `traits::eq_int_type(traits::to_int_type(c), delim)` for the next available input character `c` (in which case `c` is extracted).

26 *Remarks:* The last condition will never occur if `traits::eq_int_type(delim, traits::eof())`.

27 *Returns:* `*this`.

```
int_type peek();
```

28 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, reads but does not extract the current input character.

29 *Returns:* `traits::eof()` if `good()` is `false`. Otherwise, returns `rdbuf()->sgetc()`.


```
basic_istream<charT,traits>& read(char_type* s, streamsize n);
```

30 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, if `!good()` calls `setstate(failbit)` which may throw an exception, and return. Otherwise extracts characters and stores them into successive locations of an array whose first element is designated by `s`.[323] Characters are extracted and stored until either of the following occurs:

- `n` characters are stored;
- end-of-file occurs on the input sequence (in which case the function calls `setstate(failbit | eofbit)`, which may throw `ios_base::failure` (27.5.5.4)).

31 *Returns:* `*this`.

```
streamsize readsome(char_type* s, streamsize n);
```

32 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1). After constructing a sentry object, if `!good()` calls `setstate(failbit)` which may throw an exception, and return. Otherwise extracts characters and stores them into successive locations of an array whose first element is designated by `s`. If `rdbuf()->in_avail() == -1`, calls `setstate(eofbit)` (which may throw `ios_base::failure` (27.5.5.4)), and extracts no characters;

- If `rdbuf()->in_avail() == 0`, extracts no characters
- If `rdbuf()->in_avail() > 0`, extracts `min(rdbuf()->in_avail(),n))`.

33 *Returns:* The number of characters extracted.

```
basic_istream<charT,traits>& putback(char_type c);
```

34 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1), except that the function first clears `eofbit`. After constructing a sentry object, if `!good()` calls `setstate(failbit)` which may throw an exception, and return. If `rdbuf()` is not null, calls `rdbuf->sputbackc()`. If `rdbuf()` is null, or if `sputbackc()` returns `traits::eof()`, calls `setstate(badbit)` (which may throw `ios_base::failure` (27.5.5.4)). [*Note:* This function extracts no characters, so the value returned by the next call to `gcount()` is 0. — *end note*]

35 *Returns:* `*this`.

```
basic_istream<charT,traits>& unget();
```

36 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1), except that the function first clears `eofbit`. After constructing a sentry object, if `!good()` calls `setstate(failbit)` which may throw an exception, and return. If `rdbuf()` is not null, calls `rdbuf()->sungetc()`. If `rdbuf()` is null, or if `sungetc()` returns `traits::eof()`, calls `setstate(badbit)` (which may throw `ios_base::failure` (27.5.5.4)). [*Note:* This function extracts no characters, so the value returned by the next call to `gcount()` is 0. — *end note*]

37 *Returns:* `*this`.

```
int sync();
```

323) Note that this function is not overloaded on types `signed char` and `unsigned char`.

38 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to `gcount()`. After constructing a sentry object, if `rdbuf()` is a null pointer, returns -1 . Otherwise, calls `rdbuf()->pubsync()` and, if that function returns -1 calls `setstate(badbit)` (which may throw `ios_base::failure` (27.5.5.4), and returns `-1`. Otherwise, returns zero.

```
pos_type tellg();
```

39 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to `gcount()`.

40 *Returns:* After constructing a sentry object, if `fail() != false`, returns `pos_type(-1)` to indicate failure. Otherwise, returns `rdbuf()->pubseekoff(0, cur, in)`.

```
basic_istream<charT,traits>& seekg(pos_type pos);
```

41 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1), except that the function first clears `eofbit`, it does not count the number of characters extracted, and it does not affect the value returned by subsequent calls to `gcount()`. After constructing a sentry object, if `fail() != true`, executes `rdbuf()->pubseekpos(pos, ios_base::in)`. In case of failure, the function calls `setstate(failbit)` (which may throw `ios_base::failure`).

42 *Returns:* `*this`.

```
basic_istream<charT,traits>& seekg(off_type off, ios_base::seekdir dir);
```

43 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to `gcount()`. After constructing a sentry object, if `fail() != true`, executes `rdbuf()->pubseekoff(off, dir, ios_base::in)`. In case of failure, the function calls `setstate(failbit)` (which may throw `ios_base::failure`).

44 *Returns:* `*this`.

27.7.2.4 Standard `basic_istream` manipulators [istream.manip]

```
namespace std {
  template <class charT, class traits>
    basic_istream<charT,traits>& ws(basic_istream<charT,traits>& is);
}
```

1 *Effects:* Behaves as an unformatted input function (as described in 27.7.2.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to is.gcount(). After constructing a sentry object extracts characters as long as the next available character `c` is whitespace or until there are no more characters in the sequence. Whitespace characters are distinguished with the same criterion as used by `sentry::sentry` (27.7.2.1.3). If `ws` stops extracting characters because there are no more available it sets `eofbit`, but not `failbit`.

2 *Returns:* `is`.

27.7.2.5 Class template basic_iostream [iostreamclass]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_iostream :
    public basic_istream<charT,traits>,
    public basic_ostream<charT,traits> {
  public:
    // types:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;

    // constructor/destructor
    explicit basic_iostream(basic_streambuf<charT,traits>* sb);
    virtual ~basic_iostream();

  protected:
    basic_iostream(const basic_iostream& rhs) = delete;
    basic_iostream(basic_iostream&& rhs);

    // assign/swap
    basic_iostream& operator=(const basic_iostream& rhs) = delete;
    basic_iostream& operator=(basic_iostream&& rhs);
    void swap(basic_iostream& rhs);
  };
}
```

1 The class `basic_iostream` inherits a number of functions that allow reading input and writing output to sequences controlled by a stream buffer.

27.7.2.5.1 basic_iostream constructors [iostream.cons]

```
explicit basic_iostream(basic_streambuf<charT,traits>* sb);
```

1 *Effects:* Constructs an object of class `basic_iostream`, assigning initial values to the base classes by calling `basic_istream<charT,traits>(sb)` (27.7.2.1) and `basic_ostream<charT,traits>(sb)` (27.7.3.1)

2 *Postcondition:* `rdbuf()==sb` and `gcount()==0`.

```
basic_iostream(basic_iostream&& rhs);
```

3 *Effects:* Move constructs from the rvalue `rhs` by constructing the `basic_istream` base class with `move(rhs)`.

27.7.2.5.2 basic_iostream destructor [iostream.dest]

```
virtual ~basic_iostream();
```

1 *Effects:* Destroys an object of class `basic_iostream`.

2 *Remarks:* Does not perform any operations on `rdbuf()`.

27.7.2.5.3 `basic_iostream` assign and swap [iostream.assign]

```
basic_iostream& operator=(basic_iostream&& rhs);
```

1 *Effects:* `swap(rhs)`.

```
void swap(basic_iostream& rhs);
```

2 *Effects:* Calls `basic_istream<charT, traits>::swap(rhs)`.

27.7.2.6 Rvalue stream extraction [istream.rvalue]

```
template <class charT, class traits, class T>
  basic_istream<charT, traits>&
  operator>>(basic_istream<charT, traits>&& is, T& x);
```

1 *Effects:* `is >>x`

2 *Returns:* `is`

27.7.3 Output streams [output.streams]

1 The header `<ostream>` defines a type and several function signatures that control output to a stream buffer along with a function template that inserts into stream rvalues.

27.7.3.1 Class template `basic_ostream` [ostream]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_ostream : virtual public basic_ios<charT,traits> {
  public:
    // types (inherited from basic_ios (27.5.5)):
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;

    // 27.7.3.2 Constructor/destructor:
    explicit basic_ostream(basic_streambuf<char_type,traits>* sb);
    virtual ~basic_ostream();

    // 27.7.3.4 Prefix/suffix:
    class sentry;

    // 27.7.3.6 Formatted output:
    basic_ostream<charT,traits>& operator<<(
      basic_ostream<charT,traits>& (*pf)(basic_ostream<charT,traits>&));
    basic_ostream<charT,traits>& operator<<(
      basic_ios<charT,traits>& (*pf)(basic_ios<charT,traits>&));
    basic_ostream<charT,traits>& operator<<(
      ios_base& (*pf)(ios_base&));

    basic_ostream<charT,traits>& operator<<(bool n);
    basic_ostream<charT,traits>& operator<<(short n);
    basic_ostream<charT,traits>& operator<<(unsigned short n);
    basic_ostream<charT,traits>& operator<<(int n);
    basic_ostream<charT,traits>& operator<<(unsigned int n);
```



```
  basic_ostream<charT,traits>& operator<<(long n);
  basic_ostream<charT,traits>& operator<<(unsigned long n);
  basic_ostream<charT,traits>& operator<<(long long n);
  basic_ostream<charT,traits>& operator<<(unsigned long long n);
  basic_ostream<charT,traits>& operator<<(float f);
  basic_ostream<charT,traits>& operator<<(double f);
  basic_ostream<charT,traits>& operator<<(long double f);

  basic_ostream<charT,traits>& operator<<(const void* p);
  basic_ostream<charT,traits>& operator<<(
    basic_streambuf<char_type,traits>* sb);

  // 27.7.3.7 Unformatted output:
  basic_ostream<charT,traits>& put(char_type c);
  basic_ostream<charT,traits>& write(const char_type* s, streamsize n);

  basic_ostream<charT,traits>& flush();

  // 27.7.3.5 seeks:
  pos_type tellp();
  basic_ostream<charT,traits>& seekp(pos_type);
  basic_ostream<charT,traits>& seekp(off_type, ios_base::seekdir);
protected:
  basic_ostream(const basic_ostream& rhs) = delete;
  basic_ostream(basic_ostream&& rhs);

  // 27.7.3.3 Assign/swap
  basic_ostream& operator=(const basic_ostream& rhs) = delete;
  basic_ostream& operator=(basic_ostream&& rhs);
  void swap(basic_ostream& rhs);
};

// 27.7.3.6.4 character inserters
template<class charT, class traits>
  basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>&,
                                          charT);
template<class charT, class traits>
  basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>&,
                                          char);
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>&,
                                         char);

// signed and unsigned
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>&,
                                         signed char);
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>&,
                                         unsigned char);

template<class charT, class traits>
  basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>&,
                                          const charT*);
template<class charT, class traits>
```



```
    basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>&,
                                            const char*);
  template<class traits>
    basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>&,
                                           const char*);

  // signed and unsigned
  template<class traits>
    basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>&,
                                           const signed char*);
  template<class traits>
    basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>&,
                                           const unsigned char*);
}
```

1 The class `basic_ostream` defines a number of member function signatures that assist in formatting and writing output to output sequences controlled by a stream buffer.

2 Two groups of member function signatures share common properties: the *formatted output functions* (or *inserters*) and the *unformatted output functions*. Both groups of output functions generate (or *insert*) output *characters* by actions equivalent to calling `rdbuf()->sputc(int_type)`. They may use other public members of `basic_ostream` except that they shall not invoke any virtual members of `rdbuf()` except `overflow()`, `xsputn()`, and `sync()`.

3 If one of these called functions throws an exception, then unless explicitly noted otherwise the output function sets `badbit` in error state. If `badbit` is on in `exceptions()`, the output function rethrows the exception without completing its actions, otherwise it does not throw anything and treat as an error.

27.7.3.2 `basic_ostream` constructors [ostream.cons]

```
explicit basic_ostream(basic_streambuf<charT,traits>* sb);
```

1 *Effects:* Constructs an object of class `basic_ostream`, assigning initial values to the base class by calling `basic_ios<charT,traits>::init(sb)` (27.5.5.2).

2 *Postcondition:* `rdbuf() == sb`.

```
virtual ~basic_ostream();
```

3 *Effects:* Destroys an object of class `basic_ostream`.

4 *Remarks:* Does not perform any operations on `rdbuf()`.

```
basic_ostream(basic_ostream&& rhs);
```

5 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by default constructing the base class and calling `basic_ios<charT, traits>::move(rhs)` to initialize the base class.

27.7.3.3 Class `basic_ostream` assign and swap [ostream.assign]

```
basic_ostream& operator=(basic_ostream&& rhs);
```

1 *Effects:* `swap(rhs)`.

2 *Returns:* `*this`.

```
void swap(basic_ostream& rhs);
```

3 *Effects:* Calls `basic_ios<charT, traits>::swap(rhs)`.

27.7.3.4 Class `basic_ostream::sentry` [ostream::sentry]

```
namespace std {
  template <class charT,class traits = char_traits<charT> >
  class basic_ostream<charT,traits>::sentry {
    bool ok_; // exposition only
  public:
    explicit sentry(basic_ostream<charT,traits>& os);
    ~sentry();
    explicit operator bool() const { return ok_; }

    sentry(const sentry&) = delete;
    sentry& operator=(const sentry&) = delete;
  };
}
```

1 The class `sentry` defines a class that is responsible for doing exception safe prefix and suffix operations.

`explicit sentry(basic_ostream<charT,traits>& os);`

2 If `os.good()` is nonzero, prepares for formatted or unformatted output. If `os.tie()` is not a null pointer, calls `os.tie()->flush()`.[324]

3 If, after any preparation is completed, `os.good()` is `true`, `ok_ == true` otherwise, `ok_ == false`. During preparation, the constructor may call `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4))[325]

`~sentry();`

4 If `((os.flags() & ios_base::unitbuf) && !uncaught_exception() && os.good())` is `true`, calls `os.rdbuf()->pubsync()`. If that function returns -1, sets `badbit` in `os.rdstate()` without propagating an exception.

`explicit operator bool() const;`

5 *Effects:* Returns `ok_`.

27.7.3.5 `basic_ostream` seek members [ostream.seeks]

1 Each seek member function begins execution by constructing an object of class `sentry`. It returns by destroying the `sentry` object.

`pos_type tellp();`

2 *Returns:* If `fail() != false`, returns `pos_type(-1)` to indicate failure. Otherwise, returns `rdbuf()->pubseekoff(0, cur, out)`.

`basic_ostream<charT,traits>& seekp(pos_type pos);`

3 *Effects:* If `fail() != true`, executes `rdbuf()->pubseekpos(pos, ios_base::out)`. In case of failure, the function calls `setstate(failbit)` (which may throw `ios_base::failure`).

4 *Returns:* `*this`.

`basic_ostream<charT,traits>& seekp(off_type off, ios_base::seekdir dir);`

324) The call `os.tie()->flush()` does not necessarily occur if the function can determine that no synchronization is necessary.
325) The `sentry` constructor and destructor can also perform additional implementation-dependent operations.

5 *Effects:* If `fail() != true`, executes `rdbuf()->pubseekoff(off, dir, ios_base::out)`. In case of failure, the function calls `setstate(failbit)` (which may throw `ios_base::failure`).

6 *Returns:* `*this`.

27.7.3.6 Formatted output functions [ostream.formatted]

27.7.3.6.1 Common requirements [ostream.formatted.reqmts]

1 Each formatted output function begins execution by constructing an object of class `sentry`. If this object returns `true` when converted to a value of type `bool`, the function endeavors to generate the requested output. If the generation fails, then the formatted output function does `setstate(ios_base::failbit)`, which might throw an exception. If an exception is thrown during output, then `ios::badbit` is turned on[326] in `*this`'s error state. If `(exceptions()&badbit) != 0` then the exception is rethrown. Whether or not an exception is thrown, the `sentry` object is destroyed before leaving the formatted output function. If no exception is thrown, the result of the formatted output function is `*this`.

2 The descriptions of the individual formatted output functions describe how they perform output and do not mention the `sentry` object.

3 If a formatted output function of a stream `os` determines padding, it does so as follows. Given a `charT` character sequence `seq` where `charT` is the character type of the stream, if the length of `seq` is less than `os.width()`, then enough copies of `os.fill()` are added to this sequence as necessary to pad to a width of `os.width()` characters. If `(os.flags() & ios_base::adjustfield) == ios_base::left` is `true`, the fill characters are placed after the character sequence; otherwise, they are placed before the character sequence.

27.7.3.6.2 Arithmetic inserters [ostream.inserters.arithmetic]

```
operator<<(bool val);
operator<<(short val);
operator<<(unsigned short val);
operator<<(int val);
operator<<(unsigned int val);
operator<<(long val);
operator<<(unsigned long val);
operator<<(long long val);
operator<<(unsigned long long val);
operator<<(float val);
operator<<(double val);
operator<<(long double val);
operator<<(const void* val);
```

1 *Effects:* The classes `num_get<>` and `num_put<>` handle locale-dependent numeric formatting and parsing. These inserter functions use the imbued `locale` value to perform numeric formatting. When `val` is of type `bool`, `long`, `unsigned long`, `long long`, `unsigned long long`, `double`, `long double`, or `const void*`, the formatting conversion occurs as if it performed the following code fragment:

```
bool failed = use_facet<
  num_put<charT,ostreambuf_iterator<charT,traits> >
    >(getloc()).put(*this, *this, fill(), val).failed();
```

When `val` is of type `short` the formatting conversion occurs as if it performed the following code fragment:

```
ios_base::fmtflags baseflags = ios_base::flags() & ios_base::basefield;
bool failed = use_facet<
  num_put<charT,ostreambuf_iterator<charT,traits> >
    >(getloc()).put(*this, *this, fill(),
```

326) without causing an `ios::failure` to be thrown.


```
    baseflags == ios_base::oct || baseflags == ios_base::hex
      ? static_cast<long>(static_cast<unsigned short>(val))
      : static_cast<long>(val)).failed();
```

When `val` is of type `int` the formatting conversion occurs as if it performed the following code fragment:

```
ios_base::fmtflags baseflags = ios_base::flags() & ios_base::basefield;
bool failed = use_facet<
  num_put<charT,ostreambuf_iterator<charT,traits> >
    >(getloc()).put(*this, *this, fill(),
    baseflags == ios_base::oct || baseflags == ios_base::hex
      ? static_cast<long>(static_cast<unsigned int>(val))
      : static_cast<long>(val)).failed();
```

When `val` is of type `unsigned short` or `unsigned int` the formatting conversion occurs as if it performed the following code fragment:

```
bool failed = use_facet<
  num_put<charT,ostreambuf_iterator<charT,traits> >
    >(getloc()).put(*this, *this, fill(),
      static_cast<unsigned long>(val)).failed();
```

When `val` is of type `float` the formatting conversion occurs as if it performed the following code fragment:

```
bool failed = use_facet<
  num_put<charT,ostreambuf_iterator<charT,traits> >
    >(getloc()).put(*this, *this, fill(),
      static_cast<double>(val)).failed();
```

2 The first argument provides an object of the `ostreambuf_iterator<>` class which is an iterator for class `basic_ostream<>`. It bypasses `ostream`s and uses `streambuf`s directly. Class `locale` relies on these types as its interface to iostreams, since for flexibility it has been abstracted away from direct dependence on `ostream`. The second parameter is a reference to the base subobject of type `ios_base`. It provides formatting specifications such as field width, and a locale from which to obtain other facets. If `failed` is `true` then does `setstate(badbit)`, which may throw an exception, and returns.

3 *Returns:* `*this`.

27.7.3.6.3 `basic_ostream::operator<<` [ostream.inserters]

```
basic_ostream<charT,traits>& operator<<
    (basic_ostream<charT,traits>& (*pf)(basic_ostream<charT,traits>&))
```

1 *Effects:* None. Does not behave as a formatted output function (as described in 27.7.3.6.1).

2 *Returns:* `pf(*this)`.[327]

```
basic_ostream<charT,traits>& operator<<
    (basic_ios<charT,traits>& (*pf)(basic_ios<charT,traits>&))
```

3 *Effects:* Calls `pf(*this)`. This inserter does not behave as a formatted output function (as described in 27.7.3.6.1).

4 *Returns:* `*this`.[328]

327) See, for example, the function signature `endl(basic_ostream&)` (27.7.3.8).
328) See, for example, the function signature `dec(ios_base&)` (27.5.6.3).


```
basic_ostream<charT,traits>& operator<<
    (ios_base& (*pf)(ios_base&))
```

5 *Effects:* Calls `pf(*this)`. This inserter does not behave as a formatted output function (as described in 27.7.3.6.1).

6 *Returns:* `*this`.

```
basic_ostream<charT,traits>& operator<<
    (basic_streambuf<charT,traits>* sb);
```

7 *Effects:* Behaves as an unformatted output function (as described in 27.7.3.7, paragraph 1). After the sentry object is constructed, if `sb` is null calls `setstate(badbit)` (which may throw `ios_base::failure`).

8 Gets characters from `sb` and inserts them in `*this`. Characters are read from `sb` and inserted until any of the following occurs:

— end-of-file occurs on the input sequence;

— inserting in the output sequence fails (in which case the character to be inserted is not extracted);

— an exception occurs while getting a character from `sb`.

9 If the function inserts no characters, it calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)). If an exception was thrown while extracting a character, the function sets `failbit` in error state, and if `failbit` is on in `exceptions()` the caught exception is rethrown.

10 *Returns:* `*this`.

27.7.3.6.4 Character inserter function templates [ostream.inserters.character]

```
template<class charT, class traits>
  basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>& out,
                                          charT c);
template<class charT, class traits>
  basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>& out,
                                          char c);
  // specialization
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>& out,
                                         char c);
  // signed and unsigned
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>& out,
                                         signed char c);
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>& out,
                                         unsigned char c);
```

1 *Effects:* Behaves as a formatted output function (27.7.3.6.1) of `out`. Constructs a character sequence `seq`. If `c` has type `char` and the character type of the stream is not `char`, then `seq` consists of `out.widen(c)`; otherwise `seq` consists of `c`. Determines padding for `seq` as described in 27.7.3.6.1. Inserts `seq` into `out`. Calls `os.width(0)`.

2 *Returns:* `out`.


```
template<class charT, class traits>
  basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>& out,
                                          const charT* s);
template<class charT, class traits>
  basic_ostream<charT,traits>& operator<<(basic_ostream<charT,traits>& out,
                                          const char* s);
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>& out,
                                         const char* s);
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>& out,
                                         const signed char* s);
template<class traits>
  basic_ostream<char,traits>& operator<<(basic_ostream<char,traits>& out,
                                         const unsigned char* s);
```

3 *Requires:* `s` shall not be a null pointer.

4 *Effects:* Behaves like a formatted inserter (as described in 27.7.3.6.1) of `out`. Creates a character sequence `seq` of `n` characters starting at `s`, each widened using `out.widen()` (27.5.5.3), where `n` is the number that would be computed as if by:

- `traits::length(s)` for the overload where the first argument is of type `basic_ostream<charT, traits>&` and the second is of type `const charT*`, and also for the overload where the first argument is of type `basic_ostream<char, traits>&` and the second is of type `const char*`,
- `std::char_traits<char>::length(s)` for the overload where the first argument is of type `basic_ostream<charT, traits>&` and the second is of type `const char*`,
- `traits::length(reinterpret_cast<const char*>(s))` for the other two overloads.

Determines padding for `seq` as described in 27.7.3.6.1. Inserts `seq` into `out`. Calls `width(0)`.

5 *Returns:* `out`.

27.7.3.7 Unformatted output functions [ostream.unformatted]

1 Each unformatted output function begins execution by constructing an object of class `sentry`. If this object returns `true`, while converting to a value of type `bool`, the function endeavors to generate the requested output. If an exception is thrown during output, then `ios::badbit` is turned on[329] in `*this`'s error state. If `(exceptions() & badbit) != 0` then the exception is rethrown. In any case, the unformatted output function ends by destroying the sentry object, then, if no exception was thrown, returning the value specified for the unformatted output function.

```
basic_ostream<charT,traits>& put(char_type c);
```

2 *Effects:* Behaves as an unformatted output function (as described in 27.7.3.7, paragraph 1). After constructing a sentry object, inserts the character `c`, if possible.[330]

3 Otherwise, calls `setstate(badbit)` (which may throw `ios_base::failure` (27.5.5.4)).

4 *Returns:* `*this`.

```
basic_ostream& write(const char_type* s, streamsize n);
```

5 *Effects:* Behaves as an unformatted output function (as described in 27.7.3.7, paragraph 1). After constructing a sentry object, obtains characters to insert from successive locations of an array whose first element is designated by `s`.[331] Characters are inserted until either of the following occurs:

329) without causing an `ios::failure` to be thrown.
330) Note that this function is not overloaded on types `signed char` and `unsigned char`.
331) Note that this function is not overloaded on types `signed char` and `unsigned char`.

— `n` characters are inserted;

— inserting in the output sequence fails (in which case the function calls `setstate(badbit)`, which may throw `ios_base::failure` (27.5.5.4)).

6 *Returns:* `*this`.

```
basic_ostream& flush();
```

7 *Effects:* Behaves as an unformatted output function (as described in 27.7.3.6.1, paragraph 1). If `rdbuf()` is not a null pointer, constructs a sentry object. If this object returns `true` when converted to a value of type `bool` the function calls `rdbuf()->pubsync()`. If that function returns -1 calls `setstate(badbit)` (which may throw `ios_base::failure` (27.5.5.4)). Otherwise, if the sentry object returns `false`, does nothing.

8 *Returns:* `*this`.

27.7.3.8 Standard `basic_ostream` manipulators [ostream.manip]

```
namespace std {
  template <class charT, class traits>
    basic_ostream<charT,traits>& endl(basic_ostream<charT,traits>& os);
}
```

1 *Effects:* Calls `os.put(os.widen('\n'))`, then `os.flush()`.

2 *Returns:* `os`.

```
namespace std {
  template <class charT, class traits>
    basic_ostream<charT,traits>& ends(basic_ostream<charT,traits>& os);
}
```

3 *Effects:* Inserts a null character into the output sequence: calls `os.put(charT())`.

4 *Returns:* `os`.

```
namespace std {
  template <class charT, class traits>
    basic_ostream<charT,traits>& flush(basic_ostream<charT,traits>& os);
}
```

5 *Effects:* Calls `os.flush()`.

6 *Returns:* `os`.

27.7.3.9 Rvalue stream insertion [ostream.rvalue]

```
template <class charT, class traits, class T>
  basic_ostream<charT, traits>&
  operator<<(basic_ostream<charT, traits>&& os, const T& x);
```

1 *Effects:* `os << x`

2 *Returns:* `os`

27.7.4 Standard manipulators [std.manip]

1 The header <iomanip> defines several functions that support extractors and inserters that alter information maintained by class ios_base and its derived classes.

unspecified resetiosflags(ios_base::fmtflags mask);

2 *Returns:* An object of unspecified type such that if out is an object of type basic_ostream<charT, traits> then the expression out << resetiosflags(mask) behaves as if it called f(out, mask), or if in is an object of type basic_istream<charT, traits> then the expression in >> resetiosflags(mask) behaves as if it called f(in, mask), where the function f is defined as:[332]

```
void f(ios_base& str, ios_base::fmtflags mask) {
  // reset specified flags
  str.setf(ios_base::fmtflags(0), mask);
}
```

The expression out << resetiosflags(mask) shall have type basic_ostream<charT,traits>& and value out. The expression in >> resetiosflags(mask) shall have type basic_istream<charT, traits>& and value in.

unspecified setiosflags(ios_base::fmtflags mask);

3 *Returns:* An object of unspecified type such that if out is an object of type basic_ostream<charT, traits> then the expression out << setiosflags(mask) behaves as if it called f(out, mask), or if in is an object of type basic_istream<charT, traits> then the expression in >> setiosflags(mask) behaves as if it called f(in, mask), where the function f is defined as:

```
void f(ios_base& str, ios_base::fmtflags mask) {
  // set specified flags
  str.setf(mask);
}
```

The expression out << setiosflags(mask) shall have type basic_ostream<charT, traits>& and value out. The expression in >> setiosflags(mask) shall have type basic_istream<charT, traits>& and value in.

unspecified setbase(int base);

4 *Returns:* An object of unspecified type such that if out is an object of type basic_ostream<charT, traits> then the expression out << setbase(base) behaves as if it called f(out, base), or if in is an object of type basic_istream<charT, traits> then the expression in >> setbase(base) behaves as if it called f(in, base), where the function f is defined as:

```
void f(ios_base& str, int base) {
  // set basefield
  str.setf(base ==  8 ? ios_base::oct :
      base == 10 ? ios_base::dec :
      base == 16 ? ios_base::hex :
      ios_base::fmtflags(0), ios_base::basefield);
}
```

332) The expression cin >>resetiosflags(ios_base::skipws) clears ios_base::skipws in the format flags stored in the basic_istream<charT,traits> object cin (the same as cin >>noskipws), and the expression cout <<resetiosflags(ios_base::showbase) clears ios_base::showbase in the format flags stored in the basic_ostream<charT,traits> object cout (the same as cout <<noshowbase).

The expression `out << setbase(base)` shall have type `basic_ostream<charT, traits>&` and value `out`. The expression `in >> setbase(base)` shall have type `basic_istream<charT, traits>&` and value `in`.

unspecified `setfill(char_type c);`

5 *Returns:* An object of unspecified type such that if `out` is an object of type `basic_ostream<charT, traits>` and `c` has type `charT` then the expression `out << setfill(c)` behaves as if it called `f(out, c)`, where the function `f` is defined as:

```
template<class charT, class traits>
void f(basic_ios<charT,traits>& str, charT c) {
  // set fill character
  str.fill(c);
}
```

The expression `out << setfill(c)` shall have type `basic_ostream<charT, traits>&` and value `out`.

unspecified `setprecision(int n);`

6 *Returns:* An object of unspecified type such that if `out` is an object of type `basic_ostream<charT, traits>` then the expression `out << setprecision(n)` behaves as if it called `f(out, n)`, or if `in` is an object of type `basic_istream<charT, traits>` then the expression `in >> setprecision(n)` behaves as if it called `f(in, n)`, where the function `f` is defined as:

```
void f(ios_base& str, int n) {
  // set precision
  str.precision(n);
}
```

The expression `out << setprecision(n)` shall have type `basic_ostream<charT, traits>&` and value `out`. The expression `in >> setprecision(n)` shall have type `basic_istream<charT, traits>&` and value `in`.

unspecified `setw(int n);`

7 *Returns:* An object of unspecified type such that if `out` is an instance of `basic_ostream<charT, traits>` then the expression `out << setw(n)` behaves as if it called `f(out, n)`, or if `in` is an object of type `basic_istream<charT, traits>` then the expression `in >> setw(n)` behaves as if it called `f(in, n)`, where the function `f` is defined as:

```
void f(ios_base& str, int n) {
  // set width
  str.width(n);
}
```

The expression `out << setw(n)` shall have type `basic_ostream<charT, traits>&` and value `out`. The expression `in >> setw(n)` shall have type `basic_istream<charT, traits>&` and value `in`.

27.7.5 Extended manipulators [ext.manip]

1 The header `<iomanip>` defines several functions that support extractors and inserters that allow for the parsing and formatting of sequences and values for money and time.

```
template <class moneyT> unspecified  get_money(moneyT& mon, bool intl = false);
```

2 *Requires:* The type `moneyT` shall be either `long double` or a specialization of the `basic_string` template (Clause 21).

3 *Effects:* The expression `in >>get_money(mon, intl)` described below behaves as a formatted input function (27.7.2.2.1).

4 *Returns:* An object of unspecified type such that if `in` is an object of type `basic_istream<charT, traits>` then the expression `in >> get_money(mon, intl)` behaves as if it called `f(in, mon, intl)`, where the function `f` is defined as:

```
template <class charT, class traits, class moneyT>
void f(basic_ios<charT, traits>& str, moneyT& mon, bool intl) {
  typedef istreambuf_iterator<charT, traits> Iter;
  typedef money_get<charT, Iter> MoneyGet;

  ios_base::iostate err = ios_base::goodbit;
  const MoneyGet &mg = use_facet<MoneyGet>(str.getloc());

  mg.get(Iter(str.rdbuf()), Iter(), intl, str, err, mon);

  if (ios_base::goodbit != err)
    str.setstate(err);
}
```

The expression `in >> get_money(mon, intl)` shall have type `basic_istream<charT, traits>&` and value `in`.

```
template <class moneyT> unspecified  put_money(const moneyT& mon, bool intl = false);
```

5 *Requires:* The type `moneyT` shall be either `long double` or a specialization of the `basic_string` template (Clause 21).

6 *Returns:* An object of unspecified type such that if `out` is an object of type `basic_ostream<charT, traits>` then the expression `out << put_money(mon, intl)` behaves as a formatted input function that calls `f(out, mon, intl)`, where the function `f` is defined as:

```
template <class charT, class traits, class moneyT>
void f(basic_ios<charT, traits>& str, const moneyT& mon, bool intl) {
  typedef ostreambuf_iterator<charT, traits> Iter;
  typedef money_put<charT, Iter> MoneyPut;

  const MoneyPut& mp = use_facet<MoneyPut>(str.getloc());
  const Iter end = mp.put(Iter(str.rdbuf()), intl, str, str.fill(), mon);

  if (end.failed())
    str.setstate(ios::badbit);
}
```

The expression `out << put_money(mon, intl)` shall have type `basic_ostream<charT, traits>&` and value `out`.


```
template <class charT> unspecified  get_time(struct tm* tmb, const charT* fmt);
```

7 *Requires:* The argument `tmb` shall be a valid pointer to an object of type `struct tm`, and the argument `fmt` shall be a valid pointer to an array of objects of type `charT` with `char_traits<charT>::length(fmt)` elements.

8 *Returns:* An object of unspecified type such that if `in` is an object of type `basic_istream<charT, traits>` then the expression `in >> get_time(tmb, fmt)` behaves as if it called `f(in, tmb, fmt)`, where the function `f` is defined as:

```
template <class charT, class traits>
void f(basic_ios<charT, traits>& str, struct tm* tmb, const charT* fmt) {
  typedef istreambuf_iterator<charT, traits> Iter;
  typedef time_get<charT, Iter> TimeGet;

  ios_base::iostate err = ios_base::goodbit;
  const TimeGet& tg = use_facet<TimeGet>(str.getloc());

  tg.get(Iter(str.rdbuf()), Iter(), str, err, tmb,
    fmt, fmt + traits::length(fmt));

  if (err != ios_base::goodbit)
    str.setstate(err):
}
```

The expression `in >> get_time(tmb, fmt)` shall have type `basic_istream<charT, traits>&` and value `in`.

```
template <class charT> unspecified  put_time(const struct tm* tmb, const charT* fmt);
```

9 *Requires:* The argument `tmb` shall be a valid pointer to an object of type `struct tm`, and the argument `fmt` shall be a valid pointer to an array of objects of type `charT` with `char_traits<charT>::length(fmt)` elements.

10 *Returns:* An object of unspecified type such that if `out` is an object of type `basic_ostream<charT, traits>` then the expression `out << put_time(tmb, fmt)` behaves as if it called `f(out, tmb, fmt)`, where the function `f` is defined as:

```
template <class charT, class traits>
void f(basic_ios<charT, traits>& str, const struct tm* tmb, const charT* fmt) {
  typedef ostreambuf_iterator<charT, traits> Iter;
  typedef time_put<charT, Iter> TimePut;

  const TimePut& tp = use_facet<TimePut>(str.getloc());
  const Iter end = tp.put(Iter(str.rdbuf()), str, str.fill(), tmb,
    fmt, fmt + traits::length(fmt));

  if (end.failed())
    str.setstate(ios_base::badbit);
}
```

The expression `out << put_time(tmb, fmt)` shall have type `basic_ostream<charT, traits>&` and value `out`.

27.7.6 Quoted manipulators [quoted.manip]

1 [*Note:* Quoted manipulators provide string insertion and extraction of quoted strings (for example, XML and CSV formats). Quoted manipulators are useful in ensuring that the content of a string with embedded spaces remains unchanged if inserted and then extracted via stream I/O. — *end note*]

```
template <class charT>
  unspecified  quoted(const charT* s, charT delim=charT('"'), charT escape=charT('\\'));
template <class charT, class traits, class Allocator>
  unspecified  quoted(const basic_string<charT, traits, Allocator>& s,
                      charT delim=charT('"'), charT escape=charT('\\'));
```

2 *Returns:* An object of unspecified type such that if `out` is an instance of `basic_ostream` with member type `char_type` the same as `charT` and with member type `traits_type`, which in the second form is the same as `traits`, then the expression `out << quoted(s, delim, escape)` behaves as a formatted output function (27.7.3.6.1) of `out`. This forms a character sequence `seq`, initially consisting of the following elements:

- `delim`.
- Each character in `s`. If the character to be output is equal to `escape` or `delim`, as determined by `traits_type::eq`, first output `escape`.
- `delim`.

Let `x` be the number of elements initially in `seq`. Then padding is determined for `seq` as described in 27.7.3.6.1, `seq` is inserted as if by calling `out.rdbuf()->sputn(seq, n)`, where `n` is the larger of `out.width()` and `x`, and `out.width(0)` is called. The expression `out << quoted(s, delim, escape)` shall have type `basic_ostream<charT, traits>&` and value `out`.

```
template <class charT, class traits, class Allocator>
  unspecified  quoted(basic_string<charT, traits, Allocator>& s,
                      charT delim=charT('"'), charT escape=charT('\\'));
```

3 *Returns:* An object of unspecified type such that:

- If `in` is an instance of `basic_istream` with member types `char_type` and `traits_type` the same as `charT` and `traits`, respectively, then the expression `in >> quoted(s, delim, escape)` behaves as if it extracts the following characters from `in` using `basic_istream::operator>>` (27.7.2.2.3) which may throw `ios_base::failure` (27.5.3.1.1):
  - If the first character extracted is equal to `delim`, as determined by `traits_type::eq`, then:
    - Turn off the `skipws` flag.
    - `s.clear()`
    - Until an unescaped `delim` character is reached or `!in`, extract characters from `in` and append them to `s`, except that if an `escape` is reached, ignore it and append the next character to `s`.
    - Discard the final `delim` character.
    - Restore the `skipws` flag to its original value.
  - Otherwise, `in >> s`.
- If `out` is an instance of `basic_ostream` with member types `char_type` and `traits_type` the same as `charT` and `traits`, respectively, then the expression `out << quoted(s, delim, escape)` behaves as specified for the `const basic_string<charT, traits, Allocator>&` overload of the `quoted` function.

The expression `in >> quoted(s, delim, escape)` shall have type `basic_istream<charT, traits>&` and value `in`. The expression `out << quoted(s, delim, escape)` shall have type `basic_ostream <charT, traits>&` and value `out`.

27.8 String-based streams [string.streams]

27.8.1 Overview [string.streams.overview]

1 The header `<sstream>` defines four class templates and eight types that associate stream buffers with objects of class `basic_string`, as described in 21.3.

Header `<sstream>` synopsis

```
namespace std {
  template <class charT, class traits = char_traits<charT>,
        class Allocator = allocator<charT> >
    class basic_stringbuf;

  typedef basic_stringbuf<char>     stringbuf;
  typedef basic_stringbuf<wchar_t> wstringbuf;

  template <class charT, class traits = char_traits<charT>,
        class Allocator = allocator<charT> >
    class basic_istringstream;

  typedef basic_istringstream<char>     istringstream;
  typedef basic_istringstream<wchar_t> wistringstream;

  template <class charT, class traits = char_traits<charT>,
        class Allocator = allocator<charT> >
    class basic_ostringstream;
  typedef basic_ostringstream<char>     ostringstream;
  typedef basic_ostringstream<wchar_t> wostringstream;

  template <class charT, class traits = char_traits<charT>,
        class Allocator = allocator<charT> >
    class basic_stringstream;
  typedef basic_stringstream<char>     stringstream;
  typedef basic_stringstream<wchar_t> wstringstream;
}
```

27.8.2 Class template `basic_stringbuf` [stringbuf]

```
namespace std {
  template <class charT, class traits = char_traits<charT>,
      class Allocator = allocator<charT> >
  class basic_stringbuf : public basic_streambuf<charT,traits> {
  public:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;
    typedef Allocator                 allocator_type;

    // 27.8.2.1 Constructors:
    explicit basic_stringbuf(ios_base::openmode which
                             = ios_base::in | ios_base::out);
```



```
    explicit basic_stringbuf
    (const basic_string<charT,traits,Allocator>& str,
     ios_base::openmode which = ios_base::in | ios_base::out);
    basic_stringbuf(const basic_stringbuf& rhs) = delete;
    basic_stringbuf(basic_stringbuf&& rhs);

    // 27.8.2.2 Assign and swap:
    basic_stringbuf& operator=(const basic_stringbuf& rhs) = delete;
    basic_stringbuf& operator=(basic_stringbuf&& rhs);
    void swap(basic_stringbuf& rhs);

    // 27.8.2.3 Get and set:
    basic_string<charT,traits,Allocator> str() const;
    void str(const basic_string<charT,traits,Allocator>& s);

  protected:
    // 27.8.2.4 Overridden virtual functions:
    virtual int_type   underflow();
    virtual int_type   pbackfail(int_type c = traits::eof());
    virtual int_type   overflow (int_type c = traits::eof());
    virtual  basic_streambuf<charT,traits>* setbuf(charT*, streamsize);


    virtual pos_type seekoff(off_type off, ios_base::seekdir way,
                             ios_base::openmode which
                               = ios_base::in | ios_base::out);
    virtual pos_type seekpos(pos_type sp,
                             ios_base::openmode which
                               = ios_base::in | ios_base::out);

  private:
    ios_base::openmode mode;  // exposition only
  };

  template <class charT, class traits, class Allocator>
  void swap(basic_stringbuf<charT, traits, Allocator>& x,
            basic_stringbuf<charT, traits, Allocator>& y);
}
```

1 The class `basic_stringbuf` is derived from `basic_streambuf` to associate possibly the input sequence and possibly the output sequence with a sequence of arbitrary *characters*. The sequence can be initialized from, or made available as, an object of class `basic_string`.

2 For the sake of exposition, the maintained data is presented here as:

— `ios_base::openmode mode`, has `in` set if the input sequence can be read, and `out` set if the output sequence can be written.

27.8.2.1 `basic_stringbuf` constructors [stringbuf.cons]

```
explicit basic_stringbuf(ios_base::openmode which =
                         ios_base::in | ios_base::out);
```

1 *Effects:* Constructs an object of class `basic_stringbuf`, initializing the base class with `basic_streambuf()` (27.6.3.1), and initializing `mode` with `which`.

2 *Postcondition:* `str() == ""`.


```
explicit basic_stringbuf(const basic_string<charT,traits,Allocator>& s,
                         ios_base::openmode which = ios_base::in | ios_base::out);
```

3 *Effects:* Constructs an object of class `basic_stringbuf`, initializing the base class with `basic_streambuf()` (27.6.3.1), and initializing `mode` with `which`. Then calls `str(s)`.

```
basic_stringbuf(basic_stringbuf&& rhs);
```

4 *Effects:* Move constructs from the rvalue `rhs`. It is implementation-defined whether the sequence pointers in `*this` (`eback()`, `gptr()`, `egptr()`, `pbase()`, `pptr()`, `epptr()`) obtain the values which `rhs` had. Whether they do or not, `*this` and `rhs` reference separate buffers (if any at all) after the construction. The openmode, locale and any other state of `rhs` is also copied.

5 *Postconditions:* Let `rhs_p` refer to the state of `rhs` just prior to this construction and let `rhs_a` refer to the state of `rhs` just after this construction.

- `str() == rhs_p.str()`
- `gptr() - eback() == rhs_p.gptr() - rhs_p.eback()`
- `egptr() - eback() == rhs_p.egptr() - rhs_p.eback()`
- `pptr() - pbase() == rhs_p.pptr() - rhs_p.pbase()`
- `epptr() - pbase() == rhs_p.epptr() - rhs_p.pbase()`
- `if (eback()) eback() != rhs_a.eback()`
- `if (gptr()) gptr() != rhs_a.gptr()`
- `if (egptr()) egptr() != rhs_a.egptr()`
- `if (pbase()) pbase() != rhs_a.pbase()`
- `if (pptr()) pptr() != rhs_a.pptr()`
- `if (epptr()) epptr() != rhs_a.epptr()`

27.8.2.2 Assign and swap [stringbuf.assign]

```
basic_stringbuf& operator=(basic_stringbuf&& rhs);
```

1 *Effects:* After the move assignment `*this` has the observable state it would have had if it had been move constructed from `rhs` (see 27.8.2.1).

2 *Returns:* `*this`.

```
void swap(basic_stringbuf& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs`.

```
template <class charT, class traits, class Allocator>
void swap(basic_stringbuf<charT, traits, Allocator>& x,
          basic_stringbuf<charT, traits, Allocator>& y);
```

4 *Effects:* `x.swap(y)`.

27.8.2.3 Member functions [stringbuf.members]

```
basic_string<charT,traits,Allocator> str() const;
```

1 *Returns:* A `basic_string` object whose content is equal to the `basic_stringbuf` underlying character sequence. If the `basic_stringbuf` was created only in input mode, the resultant `basic_string` contains the character sequence in the range `[eback(),egptr())`. If the `basic_stringbuf` was created with `which & ios_base::out` being true then the resultant `basic_string` contains the character sequence in the range `[pbase(),high_mark)`, where `high_mark` represents the position one past the highest initialized character in the buffer. Characters can be initialized by writing to the stream, by constructing the `basic_stringbuf` with a `basic_string`, or by calling the `str(basic_string)` member function. In the case of calling the `str(basic_string)` member function, all characters initialized prior to the call are now considered uninitialized (except for those characters re-initialized by the new `basic_string`). Otherwise the `basic_stringbuf` has been created in neither input nor output mode and a zero length `basic_string` is returned.

```
void str(const basic_string<charT,traits,Allocator>& s);
```

2 *Effects:* Copies the content of `s` into the `basic_stringbuf` underlying character sequence and initializes the input and output sequences according to `mode`.

3 *Postconditions:* If `mode & ios_base::out` is true, `pbase()` points to the first underlying character and `epptr() >= pbase() + s.size()` holds; in addition, if `mode & ios_base::ate` is true, `pptr() == pbase() + s.size()` holds, otherwise `pptr() == pbase()` is true. If `mode & ios_base::in` is true, `eback()` points to the first underlying character, and both `gptr() == eback()` and `egptr() == eback() + s.size()` hold.

27.8.2.4 Overridden virtual functions [stringbuf.virtuals]

```
int_type underflow();
```

1 *Returns:* If the input sequence has a read position available, returns `traits::to_int_type(*gptr())`. Otherwise, returns `traits::eof()`. Any character in the underlying buffer which has been initialized is considered to be part of the input sequence.

```
int_type pbackfail(int_type c = traits::eof());
```

2 *Effects:* Puts back the character designated by `c` to the input sequence, if possible, in one of three ways:

- If `traits::eq_int_type(c,traits::eof())` returns `false` and if the input sequence has a putback position available, and if `traits::eq(to_char_type(c),gptr()[-1])` returns `true`, assigns `gptr() - 1` to `gptr()`.
  Returns: `c`.
- If `traits::eq_int_type(c,traits::eof())` returns `false` and if the input sequence has a putback position available, and if `mode & ios_base::out` is nonzero, assigns `c` to `*--gptr()`.
  Returns: `c`.
- If `traits::eq_int_type(c,traits::eof())` returns `true` and if the input sequence has a putback position available, assigns `gptr() - 1` to `gptr()`.
  Returns: `traits::not_eof(c)`.

3 *Returns:* `traits::eof()` to indicate failure.

4 *Remarks:* If the function can succeed in more than one of these ways, it is unspecified which way is chosen.


```
int_type overflow(int_type c = traits::eof());
```

Effects: Appends the character designated by `c` to the output sequence, if possible, in one of two ways:

— If `traits::eq_int_type(c,traits::eof())` returns `false` and if either the output sequence has a write position available or the function makes a write position available (as described below), the function calls `sputc(c )`.
Signals success by returning `c`.

— If `traits::eq_int_type(c,traits::eof())` returns `true`, there is no character to append.
Signals success by returning a value other than `traits::eof()`.

Remarks: The function can alter the number of write positions available as a result of any call.

Returns: `traits::eof()` to indicate failure.

The function can make a write position available only if `(mode & ios_base::out) != 0`. To make a write position available, the function reallocates (or initially allocates) an array object with a sufficient number of elements to hold the current array object (if any), plus at least one additional write position. If `(mode & ios_base::in) != 0`, the function alters the read end pointer `egptr()` to point just past the new write position.

```
pos_type seekoff(off_type off, ios_base::seekdir way,
                 ios_base::openmode which
                   = ios_base::in | ios_base::out);
```

Effects: Alters the stream position within one of the controlled sequences, if possible, as indicated in Table 130.

Table 130 — `seekoff` positioning

Conditions	Result
`(which & ios_base::in) == ios_base::in`	positions the input sequence
`(which & ios_base::out) == ios_base::out`	positions the output sequence
`(which & (ios_base::in | ios_base::out)) == (ios_base::in | ios_base::out))` and `way ==` either `ios_base::beg` or `ios_base::end`	positions both the input and the output sequences
Otherwise	the positioning operation fails.

For a sequence to be positioned, if its next pointer (either `gptr()` or `pptr()`) is a null pointer and the new offset `newoff` is nonzero, the positioning operation fails. Otherwise, the function determines `newoff` as indicated in Table 131.

If `(newoff + off) < 0`, or if `newoff + off` refers to an uninitialized character (as defined in 27.8.2.3 paragraph 1), the positioning operation fails. Otherwise, the function assigns `xbeg + newoff + off` to the next pointer `xnext`.

Table 131 — `newoff` values

Condition	`newoff` Value
`way == ios_base::beg`	0
`way == ios_base::cur`	the next pointer minus the beginning pointer (`xnext - xbeg`).
`way == ios_base::end`	the high mark pointer minus the beginning pointer (`high_mark - xbeg`).

12 *Returns:* `pos_type(newoff)`, constructed from the resultant offset `newoff` (of type `off_type`), that stores the resultant stream position, if possible. If the positioning operation fails, or if the constructed object cannot represent the resultant stream position, the return value is `pos_type(off_type(-1))`.

```
pos_type seekpos(pos_type sp, ios_base::openmode which
                  = ios_base::in | ios_base::out);
```

13 *Effects:* Equivalent to `seekoff(off_type(sp), ios_base::beg, which)`.

14 *Returns:* `sp` to indicate success, or `pos_type(off_type(-1))` to indicate failure.

```
basic_streambuf<charT,traits>* setbuf(charT* s, streamsize n);
```

15 *Effects:* implementation-defined, except that `setbuf(0,0)` has no effect.

16 *Returns:* `this`.

27.8.3 Class template `basic_istringstream` [istringstream]

```
namespace std {
  template <class charT, class traits = char_traits<charT>,
        class Allocator = allocator<charT> >
  class basic_istringstream : public basic_istream<charT,traits> {
  public:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;
    typedef Allocator                 allocator_type;

    // 27.8.3.1 Constructors:
    explicit basic_istringstream(ios_base::openmode which = ios_base::in);
    explicit basic_istringstream(
               const basic_string<charT,traits,Allocator>& str,
               ios_base::openmode which = ios_base::in);
    basic_istringstream(const basic_istringstream& rhs) = delete;
    basic_istringstream(basic_istringstream&& rhs);

    // 27.8.3.2 Assign and swap:
    basic_istringstream& operator=(const basic_istringstream& rhs) = delete;
    basic_istringstream& operator=(basic_istringstream&& rhs);
```



```
    void swap(basic_istringstream& rhs);

    // 27.8.3.3 Members:
    basic_stringbuf<charT,traits,Allocator>* rdbuf() const;

    basic_string<charT,traits,Allocator> str() const;
    void str(const basic_string<charT,traits,Allocator>& s);
  private:
    basic_stringbuf<charT,traits,Allocator> sb; // exposition only
  };

  template <class charT, class traits, class Allocator>
  void swap(basic_istringstream<charT, traits, Allocator>& x,
            basic_istringstream<charT, traits, Allocator>& y);
}
```

1 The class `basic_istringstream<charT, traits, Allocator>` supports reading objects of class `basic_string<charT, traits, Allocator>`. It uses a `basic_stringbuf<charT, traits, Allocator>` object to control the associated storage. For the sake of exposition, the maintained data is presented here as:

— `sb`, the `stringbuf` object.

27.8.3.1 `basic_istringstream` constructors [istringstream.cons]

```
explicit basic_istringstream(ios_base::openmode which = ios_base::in);
```

1 *Effects:* Constructs an object of class `basic_istringstream<charT, traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(which | ios_base::in))` (27.8.2.1).

```
explicit basic_istringstream(
               const basic_string<charT, traits, Allocator>& str,
               ios_base::openmode which = ios_base::in);
```

2 *Effects:* Constructs an object of class `basic_istringstream<charT, traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::in))` (27.8.2.1).

```
basic_istringstream(basic_istringstream&& rhs);
```

3 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_stringbuf`. Next `basic_istream<charT,traits>::set_rdbuf(&sb)` is called to install the contained `basic_stringbuf`.

27.8.3.2 Assign and swap [istringstream.assign]

```
basic_istringstream& operator=(basic_istringstream&& rhs);
```

1 *Effects:* Move assigns the base and members of `*this` from the base and corresponding members of `rhs`.

2 *Returns:* `*this`.

```
void swap(basic_istringstream& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs` by calling `basic_istream<charT,traits>::swap(rhs)` and `sb.swap(rhs.sb)`.


```
template <class charT, class traits, class Allocator>
void swap(basic_istringstream<charT, traits, Allocator>& x,
          basic_istringstream<charT, traits, Allocator>& y);
```

4 *Effects:* `x.swap(y)`.

27.8.3.3 Member functions [istringstream.members]

```
basic_stringbuf<charT,traits,Allocator>* rdbuf() const;
```

1 *Returns:* `const_cast<basic_stringbuf<charT,traits,Allocator>*>(&sb)`.

```
basic_string<charT,traits,Allocator> str() const;
```

2 *Returns:* `rdbuf()->str()`.

```
void str(const basic_string<charT,traits,Allocator>& s);
```

3 *Effects:* Calls `rdbuf()->str(s)`.

27.8.4 Class template `basic_ostringstream` [ostringstream]

```
namespace std {
  template <class charT, class traits = char_traits<charT>,
        class Allocator = allocator<charT> >
  class basic_ostringstream : public basic_ostream<charT,traits> {
  public:

    // types:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;
    typedef Allocator                 allocator_type;

    // 27.8.4.1 Constructors/destructor:
    explicit basic_ostringstream(ios_base::openmode which = ios_base::out);
    explicit basic_ostringstream(
             const basic_string<charT,traits,Allocator>& str,
             ios_base::openmode which = ios_base::out);
    basic_ostringstream(const basic_ostringstream& rhs) = delete;
    basic_ostringstream(basic_ostringstream&& rhs);

    // 27.8.4.2 Assign/swap:
    basic_ostringstream& operator=(const basic_ostringstream& rhs) = delete;
    basic_ostringstream& operator=(basic_ostringstream&& rhs);
    void swap(basic_ostringstream& rhs);

    // 27.8.4.3 Members:
    basic_stringbuf<charT,traits,Allocator>* rdbuf() const;

    basic_string<charT,traits,Allocator> str() const;
    void    str(const basic_string<charT,traits,Allocator>& s);
   private:
```



```
    basic_stringbuf<charT,traits,Allocator> sb; // exposition only
  };

  template <class charT, class traits, class Allocator>
  void swap(basic_ostringstream<charT, traits, Allocator>& x,
            basic_ostringstream<charT, traits, Allocator>& y);
}
```

1 The class `basic_ostringstream<charT, traits, Allocator>` supports writing objects of class `basic_string<charT, traits, Allocator>`. It uses a `basic_stringbuf` object to control the associated storage. For the sake of exposition, the maintained data is presented here as:

— `sb`, the `stringbuf` object.

27.8.4.1 `basic_ostringstream` constructors [ostringstream.cons]

```
explicit basic_ostringstream(ios_base::openmode which = ios_base::out);
```

1 *Effects:* Constructs an object of class `basic_ostringstream`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(which | ios_base::out))` (27.8.2.1).

```
explicit basic_ostringstream(
  const basic_string<charT,traits,Allocator>& str,
  ios_base::openmode which = ios_base::out);
```

2 *Effects:* Constructs an object of class `basic_ostringstream<charT, traits>`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which | ios_base::out))` (27.8.2.1).

```
basic_ostringstream(basic_ostringstream&& rhs);
```

3 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_stringbuf`. Next `basic_ostream<charT,traits>::set_rdbuf(&sb)` is called to install the contained `basic_stringbuf`.

27.8.4.2 Assign and swap [ostringstream.assign]

```
basic_ostringstream& operator=(basic_ostringstream&& rhs);
```

1 *Effects:* Move assigns the base and members of `*this` from the base and corresponding members of `rhs`.

2 *Returns:* `*this`.

```
void swap(basic_ostringstream& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs` by calling `basic_ostream<charT,traits>::swap(rhs)` and `sb.swap(rhs.sb)`.

```
template <class charT, class traits, class Allocator>
void swap(basic_ostringstream<charT, traits, Allocator>& x,
          basic_ostringstream<charT, traits, Allocator>& y);
```

4 *Effects:* `x.swap(y)`.

27.8.4.3 Member functions [ostringstream.members]

```
basic_stringbuf<charT,traits,Allocator>* rdbuf() const;
```

1 *Returns:* `const_cast<basic_stringbuf<charT,traits,Allocator>*>(&sb)`.

```
basic_string<charT,traits,Allocator> str() const;
```

2 *Returns:* `rdbuf()->str()`.

```
void str(const basic_string<charT,traits,Allocator>& s);
```

3 *Effects:* Calls `rdbuf()->str(s)`.

27.8.5 Class template `basic_stringstream` [stringstream]

```
namespace std {
  template <class charT, class traits = char_traits<charT>,
        class Allocator = allocator<charT> >
  class basic_stringstream
    : public basic_iostream<charT,traits> {
  public:

    // types:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;
    typedef Allocator                 allocator_type;

    // constructors/destructor
    explicit basic_stringstream(
    ios_base::openmode which = ios_base::out|ios_base::in);
    explicit basic_stringstream(
    const basic_string<charT,traits,Allocator>& str,
    ios_base::openmode which = ios_base::out|ios_base::in);
    basic_stringstream(const basic_stringstream& rhs) = delete;
    basic_stringstream(basic_stringstream&& rhs);

    // 27.8.6.1 Assign/swap:
    basic_stringstream& operator=(const basic_stringstream& rhs) = delete;
    basic_stringstream& operator=(basic_stringstream&& rhs);
    void swap(basic_stringstream& rhs);

    // Members:
    basic_stringbuf<charT,traits,Allocator>* rdbuf() const;
    basic_string<charT,traits,Allocator> str() const;
    void str(const basic_string<charT,traits,Allocator>& str);

  private:
    basic_stringbuf<charT, traits> sb;  // exposition only
  };

  template <class charT, class traits, class Allocator>
```



```
    void swap(basic_stringstream<charT, traits, Allocator>& x,
              basic_stringstream<charT, traits, Allocator>& y);
}
```

1 The class template `basic_stringstream<charT, traits>` supports reading and writing from objects of class `basic_string<charT, traits, Allocator>`. It uses a `basic_stringbuf<charT, traits, Allocator>` object to control the associated sequence. For the sake of exposition, the maintained data is presented here as

— `sb`, the `stringbuf` object.

27.8.6 basic_stringstream constructors [stringstream.cons]

```
explicit basic_stringstream(
    ios_base::openmode which = ios_base::out|ios_base::in);
```

1 *Effects:* Constructs an object of class `basic_stringstream<charT,traits>`, initializing the base class with `basic_iostream(&sb)` and initializing `sb` with `basic_stringbuf<charT,traits,Allocator>(which)`.

```
explicit basic_stringstream(
    const basic_string<charT,traits,Allocator>& str,
    ios_base::openmode which = ios_base::out|ios_base::in);
```

2 *Effects:* Constructs an object of class `basic_stringstream<charT, traits>`, initializing the base class with `basic_iostream(&sb)` and initializing `sb` with `basic_stringbuf<charT, traits, Allocator>(str, which)`.

```
basic_stringstream(basic_stringstream&& rhs);
```

3 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_stringbuf`. Next `basic_istream<charT,traits>::set_rdbuf(&sb)` is called to install the contained `basic_stringbuf`.

27.8.6.1 Assign and swap [stringstream.assign]

```
basic_stringstream& operator=(basic_stringstream&& rhs);
```

1 *Effects:* Move assigns the base and members of `*this` from the base and corresponding members of `rhs`.

2 *Returns:* `*this`.

```
void swap(basic_stringstream& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs` by calling `basic_iostream<charT,traits>::swap(rhs)` and `sb.swap(rhs.sb)`.

```
template <class charT, class traits, class Allocator>
void swap(basic_stringstream<charT, traits, Allocator>& x,
          basic_stringstream<charT, traits, Allocator>& y);
```

4 *Effects:* `x.swap(y)`.

27.8.7 Member functions [stringstream.members]

```
basic_stringbuf<charT,traits,Allocator>* rdbuf() const;
```

1 *Returns:* `const_cast<basic_stringbuf<charT,traits,Allocator>*>(&sb)`

```
basic_string<charT,traits,Allocator> str() const;
```

2 *Returns:* `rdbuf()->str()`.

```
void str(const basic_string<charT,traits,Allocator>& str);
```

3 *Effects:* Calls `rdbuf()->str(str)`.

27.9 File-based streams [file.streams]

27.9.1 File streams [fstreams]

1 The header `<fstream>` defines four class templates and eight types that associate stream buffers with files and assist reading and writing files.

Header `<fstream>` synopsis

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
    class basic_filebuf;
  typedef basic_filebuf<char>    filebuf;
  typedef basic_filebuf<wchar_t> wfilebuf;

  template <class charT, class traits = char_traits<charT> >
    class basic_ifstream;
  typedef basic_ifstream<char>    ifstream;
  typedef basic_ifstream<wchar_t> wifstream;

  template <class charT, class traits = char_traits<charT> >
    class basic_ofstream;
  typedef basic_ofstream<char>    ofstream;
  typedef basic_ofstream<wchar_t> wofstream;

  template <class charT, class traits = char_traits<charT> >
    class basic_fstream;
  typedef basic_fstream<char>     fstream;
  typedef basic_fstream<wchar_t> wfstream;
}
```

2 In this subclause, the type name `FILE` refers to the type `FILE` declared in `<cstdio>` (27.9.2).

3 [*Note:* The class template `basic_filebuf` treats a file as a source or sink of bytes. In an environment that uses a large character set, the file typically holds multibyte character sequences and the `basic_filebuf` object converts those multibyte sequences into wide character sequences. — *end note*]

27.9.1.1 Class template `basic_filebuf` [filebuf]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_filebuf : public basic_streambuf<charT,traits> {
  public:
    typedef charT                    char_type;
    typedef typename traits::int_type int_type;
```



```
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;

    // 27.9.1.2 Constructors/destructor:
    basic_filebuf();
    basic_filebuf(const basic_filebuf& rhs) = delete;
    basic_filebuf(basic_filebuf&& rhs);
    virtual ~basic_filebuf();

    // 27.9.1.3 Assign/swap:
    basic_filebuf& operator=(const basic_filebuf& rhs) = delete;
    basic_filebuf& operator=(basic_filebuf&& rhs);
    void swap(basic_filebuf& rhs);

     // 27.9.1.4 Members:
    bool is_open() const;
    basic_filebuf<charT,traits>* open(const char* s,
        ios_base::openmode mode);
    basic_filebuf<charT,traits>* open(const string& s,
        ios_base::openmode mode);
    basic_filebuf<charT,traits>* close();

  protected:
    // 27.9.1.5 Overridden virtual functions:
    virtual streamsize showmanyc();
    virtual int_type underflow();
    virtual int_type uflow();
    virtual int_type pbackfail(int_type c = traits::eof());
    virtual int_type overflow (int_type c = traits::eof());

    virtual basic_streambuf<charT,traits>*
        setbuf(char_type* s, streamsize n);
    virtual pos_type seekoff(off_type off, ios_base::seekdir way,
        ios_base::openmode which = ios_base::in | ios_base::out);
    virtual pos_type seekpos(pos_type sp,
        ios_base::openmode which = ios_base::in | ios_base::out);
    virtual int      sync();
    virtual void     imbue(const locale& loc);
  };

  template <class charT, class traits>
  void swap(basic_filebuf<charT, traits>& x,
            basic_filebuf<charT, traits>& y);
}
```

1 The class `basic_filebuf<charT,traits>` associates both the input sequence and the output sequence with a file.

2 The restrictions on reading and writing a sequence controlled by an object of class `basic_filebuf<charT, traits>` are the same as for reading and writing with the Standard C library `FILE`s.

3 In particular:

— If the file is not open for reading the input sequence cannot be read.

— If the file is not open for writing the output sequence cannot be written.

— A joint file position is maintained for both the input sequence and the output sequence.

4 An instance of `basic_filebuf` behaves as described in 27.9.1.1 provided `traits::pos_type` is `fpos<traits::state_type>`. Otherwise the behavior is undefined.

5 In order to support file I/O and multibyte/wide character conversion, conversions are performed using members of a facet, referred to as `a_codecvt` in following sections, obtained as if by

```
const codecvt<charT,char,typename traits::state_type>& a_codecvt =
  use_facet<codecvt<charT,char,typename traits::state_type> >(getloc());
```

27.9.1.2 `basic_filebuf` constructors [filebuf.cons]

```
basic_filebuf();
```

1 *Effects:* Constructs an object of class `basic_filebuf<charT,traits>`, initializing the base class with `basic_streambuf<charT,traits>()` (27.6.3.1).

2 *Postcondition:* `is_open() == false`.

```
basic_filebuf(basic_filebuf&& rhs);
```

3 *Effects:* Move constructs from the rvalue `rhs`. It is implementation-defined whether the sequence pointers in `*this` (`eback()`, `gptr()`, `egptr()`, `pbase()`, `pptr()`, `epptr()`) obtain the values which `rhs` had. Whether they do or not, `*this` and `rhs` reference separate buffers (if any at all) after the construction. Additionally `*this` references the file which `rhs` did before the construction, and `rhs` references no file after the construction. The openmode, locale and any other state of `rhs` is also copied.

4 *Postconditions:* Let `rhs_p` refer to the state of `rhs` just prior to this construction and let `rhs_a` refer to the state of `rhs` just after this construction.

- `is_open() == rhs_p.is_open()`
- `rhs_a.is_open() == false`
- `gptr() - eback() == rhs_p.gptr() - rhs_p.eback()`
- `egptr() - eback() == rhs_p.egptr() - rhs_p.eback()`
- `pptr() - pbase() == rhs_p.pptr() - rhs_p.pbase()`
- `epptr() - pbase() == rhs_p.epptr() - rhs_p.pbase()`
- `if (eback()) eback() != rhs_a.eback()`
- `if (gptr()) gptr() != rhs_a.gptr()`
- `if (egptr()) egptr() != rhs_a.egptr()`
- `if (pbase()) pbase() != rhs_a.pbase()`
- `if (pptr()) pptr() != rhs_a.pptr()`
- `if (epptr()) epptr() != rhs_a.epptr()`

```
virtual ~basic_filebuf();
```

5 *Effects:* Destroys an object of class `basic_filebuf<charT,traits>`. Calls `close()`. If an exception occurs during the destruction of the object, including the call to `close()`, the exception is caught but not rethrown (see 17.6.5.12).

27.9.1.3 Assign and swap [filebuf.assign]

```
basic_filebuf& operator=(basic_filebuf&& rhs);
```

1 *Effects:* Calls `this->close()` then move assigns from `rhs`. After the move assignment `*this` has the observable state it would have had if it had been move constructed from `rhs` (see 27.9.1.2).

2 *Returns:* `*this`.

```
void swap(basic_filebuf& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs`.

```
template <class charT, class traits>
void swap(basic_filebuf<charT, traits>& x,
          basic_filebuf<charT, traits>& y);
```

4 *Effects:* `x.swap(y)`.

27.9.1.4 Member functions [filebuf.members]

```
bool is_open() const;
```

1 *Returns:* `true` if a previous call to `open` succeeded (returned a non-null value) and there has been no intervening call to close.

```
basic_filebuf<charT,traits>* open(const char* s,
    ios_base::openmode mode);
```

2 *Effects:* If `is_open() != false`, returns a null pointer. Otherwise, initializes the `filebuf` as required. It then opens a file, if possible, whose name is the NTBS `s` (as if by calling `std::fopen(s,modstr)`). The NTBS `modstr` is determined from `mode & ~ios_base::ate` as indicated in Table 132. If `mode` is not some combination of flags shown in the table then the open fails.

3 If the open operation succeeds and `(mode & ios_base::ate) != 0`, positions the file to the end (as if by calling `std::fseek(file,0,SEEK_END)`).[333]

4 If the repositioning operation fails, calls `close()` and returns a null pointer to indicate failure.

5 *Returns:* `this` if successful, a null pointer otherwise.

```
basic_filebuf<charT,traits>* open(const string& s,
    ios_base::openmode mode);
```

Returns: `open(s.c_str(), mode);`

```
basic_filebuf<charT,traits>* close();
```

333) The macro `SEEK_END` is defined, and the function signatures `fopen(const char*, const char*)` and `fseek(FILE*, long, int)` are declared, in `<cstdio>` (27.9.2).

Table 132 — File open modes

`ios_base` flag combination					`stdio` equivalent
`binary`	`in`	`out`	`trunc`	`app`	
		+			`"w"`
		+		+	`"a"`
				+	`"a"`
		+	+		`"w"`
	+				`"r"`
	+	+			`"r+"`
	+	+	+		`"w+"`
	+	+		+	`"a+"`
	+			+	`"a+"`
+		+			`"wb"`
+		+		+	`"ab"`
+				+	`"ab"`
+		+	+		`"wb"`
+	+				`"rb"`
+	+	+			`"r+b"`
+	+	+	+		`"w+b"`
+	+	+		+	`"a+b"`
+	+			+	`"a+b"`

6 *Effects:* If `is_open() == false`, returns a null pointer. If a put area exists, calls `overflow(traits::eof())` to flush characters. If the last virtual member function called on `*this` (between `underflow`, `overflow`, `seekoff`, and `seekpos`) was `overflow` then calls `a_codecvt.unshift` (possibly several times) to determine a termination sequence, inserts those characters and calls `overflow(traits::eof())` again. Finally, regardless of whether any of the preceding calls fails or throws an exception, the function closes the file (as if by calling `std::fclose(file)`).[334] If any of the calls made by the function, including `std::fclose`, fails, `close` fails by returning a null pointer. If one of these calls throws an exception, the exception is caught and rethrown after closing the file.

7 *Returns:* `this` on success, a null pointer otherwise.

8 *Postcondition:* `is_open() == false`.

27.9.1.5 Overridden virtual functions [filebuf.virtuals]

```
streamsize showmanyc();
```

1 *Effects:* Behaves the same as `basic_streambuf::showmanyc()` (27.6.3.4).

2 *Remarks:* An implementation might well provide an overriding definition for this function signature if it can determine that more characters can be read from the input sequence.

```
int_type underflow();
```

3 *Effects:* Behaves according to the description of `basic_streambuf<charT,traits>::underflow()`, with the specialization that a sequence of characters is read from the input sequence as if by reading from the associated file into an internal buffer (`extern_buf`) and then as if by doing

334) The function signature `fclose(FILE*)` is declared in `<cstdio>` (27.9.2).


```
char   extern_buf[XSIZE];
char*  extern_end;
charT  intern_buf[ISIZE];
charT* intern_end;
codecvt_base::result r =
  a_codecvt.in(state, extern_buf, extern_buf+XSIZE, extern_end,
               intern_buf, intern_buf+ISIZE, intern_end);
```

This shall be done in such a way that the class can recover the position (`fpos_t`) corresponding to each character between `intern_buf` and `intern_end`. If the value of `r` indicates that `a_codecvt.in()` ran out of space in `intern_buf`, retry with a larger `intern_buf`.

```
int_type uflow();
```

4 *Effects:* Behaves according to the description of `basic_streambuf<charT,traits>::uflow()`, with the specialization that a sequence of characters is read from the input with the same method as used by `underflow`.

```
int_type pbackfail(int_type c = traits::eof());
```

5 *Effects:* Puts back the character designated by `c` to the input sequence, if possible, in one of three ways:

- If `traits::eq_int_type(c,traits::eof())` returns `false` and if the function makes a putback position available and if `traits::eq(to_char_type(c),gptr()[-1])` returns `true`, decrements the next pointer for the input sequence, `gptr()`.

  Returns: `c`.
- If `traits::eq_int_type(c,traits::eof())` returns `false` and if the function makes a putback position available and if the function is permitted to assign to the putback position, decrements the next pointer for the input sequence, and stores `c` there.

  Returns: `c`.
- If `traits::eq_int_type(c,traits::eof())` returns `true`, and if either the input sequence has a putback position available or the function makes a putback position available, decrements the next pointer for the input sequence, `gptr()`.

  Returns: `traits::not_eof(c)`.

6 *Returns:* `traits::eof()` to indicate failure.

7 *Remarks:* If `is_open() == false`, the function always fails.

8 The function does not put back a character directly to the input sequence.

9 If the function can succeed in more than one of these ways, it is unspecified which way is chosen. The function can alter the number of putback positions available as a result of any call.

```
int_type overflow(int_type c = traits::eof());
```

10 *Effects:* Behaves according to the description of `basic_streambuf<charT,traits>::overflow(c)`, except that the behavior of "consuming characters" is performed by first converting as if by:


```
charT* b = pbase();
charT* p = pptr();
charT* end;
char   xbuf[XSIZE];
char*  xbuf_end;
codecvt_base::result r =
  a_codecvt.out(state, b, p, end, xbuf, xbuf+XSIZE, xbuf_end);
```

and then

- If `r == codecvt_base::error` then fail.
- If `r == codecvt_base::noconv` then output characters from `b` up to (and not including) `p`.
- If `r == codecvt_base::partial` then output to the file characters from `xbuf` up to `xbuf_end`, and repeat using characters from `end` to `p`. If output fails, fail (without repeating).
- Otherwise output from `xbuf` to `xbuf_end`, and fail if output fails. At this point if `b != p` and `b == end` (`xbuf` isn't large enough) then increase `XSIZE` and repeat from the beginning.

11 *Returns:* `traits::not_eof(c)` to indicate success, and `traits::eof()` to indicate failure. If `is_open() == false`, the function always fails.

```
basic_streambuf* setbuf(char_type* s, streamsize n);
```

12 *Effects:* If `setbuf(0,0)` is called on a stream before any I/O has occurred on that stream, the stream becomes unbuffered. Otherwise the results are implementation-defined. "Unbuffered" means that `pbase()` and `pptr()` always return null and output to the file should appear as soon as possible.

```
pos_type seekoff(off_type off, ios_base::seekdir way,
    ios_base::openmode which = ios_base::in | ios_base::out);
```

13 *Effects:* Let `width` denote `a_codecvt.encoding()`. If `is_open() == false`, or `off != 0 && width <= 0`, then the positioning operation fails. Otherwise, if `way != basic_ios::cur` or `off != 0`, and if the last operation was output, then update the output sequence and write any unshift sequence. Next, seek to the new position: if `width > 0`, call `std::fseek(file, width * off, whence)`, otherwise call `std::fseek(file, 0, whence)`.

14 *Remarks:* "The last operation was output" means either the last virtual operation was overflow or the put buffer is non-empty. "Write any unshift sequence" means, if `width` if less than zero then call `a_codecvt.unshift(state, xbuf, xbuf+XSIZE, xbuf_end)` and output the resulting unshift sequence. The function determines one of three values for the argument `whence`, of type `int`, as indicated in Table 133.

Table 133 — `seekoff` effects

`way` Value	`stdio` Equivalent
`basic_ios::beg`	`SEEK_SET`
`basic_ios::cur`	`SEEK_CUR`
`basic_ios::end`	`SEEK_END`

15 *Returns:* A newly constructed `pos_type` object that stores the resultant stream position, if possible. If the positioning operation fails, or if the object cannot represent the resultant stream position, returns `pos_type(off_type(-1))`.


```
pos_type seekpos(pos_type sp,
    ios_base::openmode which = ios_base::in | ios_base::out);
```

16 Alters the file position, if possible, to correspond to the position stored in `sp` (as described below). Altering the file position performs as follows:

1. if `(om & ios_base::out) != 0`, then update the output sequence and write any unshift sequence;
2. set the file position to `sp`;
3. if `(om & ios_base::in) != 0`, then update the input sequence;

where `om` is the open mode passed to the last call to `open()`. The operation fails if `is_open()` returns false.

17 If `sp` is an invalid stream position, or if the function positions neither sequence, the positioning operation fails. If `sp` has not been obtained by a previous successful call to one of the positioning functions (`seekoff` or `seekpos`) on the same file the effects are undefined.

18 *Returns:* `sp` on success. Otherwise returns `pos_type(off_type(-1))`.

```
int sync();
```

19 *Effects:* If a put area exists, calls `filebuf::overflow` to write the characters to the file. If a get area exists, the effect is implementation-defined.

```
void imbue(const locale& loc);
```

20 *Requires:* If the file is not positioned at its beginning and the encoding of the current locale as determined by `a_codecvt.encoding()` is state-dependent (22.4.1.4.2) then that facet is the same as the corresponding facet of `loc`.

21 *Effects:* Causes characters inserted or extracted after this call to be converted according to `loc` until another call of `imbue`.

22 *Remark:* This may require reconversion of previously converted characters. This in turn may require the implementation to be able to reconstruct the original contents of the file.

27.9.1.6 Class template `basic_ifstream` [ifstream]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_ifstream : public basic_istream<charT,traits> {
  public:
    typedef charT                     char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                    traits_type;

    // 27.9.1.7 Constructors:
    basic_ifstream();
    explicit basic_ifstream(const char* s,
        ios_base::openmode mode = ios_base::in);
    explicit basic_ifstream(const string& s,
        ios_base::openmode mode = ios_base::in);
    basic_ifstream(const basic_ifstream& rhs) = delete;
```



```
    basic_ifstream(basic_ifstream&& rhs);

    // 27.9.1.8 Assign/swap:
    basic_ifstream& operator=(const basic_ifstream& rhs) = delete;
    basic_ifstream& operator=(basic_ifstream&& rhs);
    void swap(basic_ifstream& rhs);

    // 27.9.1.9 Members:
    basic_filebuf<charT,traits>* rdbuf() const;

    bool is_open() const;
    void open(const char* s, ios_base::openmode mode = ios_base::in);
    void open(const string& s, ios_base::openmode mode = ios_base::in);
    void close();
  private:
    basic_filebuf<charT,traits> sb; // exposition only
  };

  template <class charT, class traits>
  void swap(basic_ifstream<charT, traits>& x,
            basic_ifstream<charT, traits>& y);
}
```

1 The class `basic_ifstream<charT, traits>` supports reading from named files. It uses a `basic_filebuf<charT, traits>` object to control the associated sequence. For the sake of exposition, the maintained data is presented here as:

— `sb`, the `filebuf` object.

27.9.1.7 `basic_ifstream` constructors [ifstream.cons]

```
basic_ifstream();
```

1 *Effects:* Constructs an object of class `basic_ifstream<charT,traits>`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_filebuf<charT,traits>()` (27.7.2.1.1, 27.9.1.2).

```
explicit basic_ifstream(const char* s,
    ios_base::openmode mode = ios_base::in);
```

2 *Effects:* Constructs an object of class `basic_ifstream`, initializing the base class with `basic_istream(&sb)` and initializing `sb` with `basic_filebuf<charT, traits>()` (27.7.2.1.1, 27.9.1.2), then calls `rdbuf()->open(s, mode | ios_base::in)`. If that function returns a null pointer, calls `setstate(failbit)`.

```
explicit basic_ifstream(const string& s,
    ios_base::openmode mode = ios_base::in);
```

3 *Effects:* the same as `basic_ifstream(s.c_str(), mode)`.

```
basic_ifstream(basic_ifstream&& rhs);
```

4 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_filebuf`. Next `basic_istream<charT,traits>::set_rdbuf(&sb)` is called to install the contained `basic_filebuf`.

27.9.1.8 Assign and swap [ifstream.assign]

```
basic_ifstream& operator=(basic_ifstream&& rhs);
```

1 *Effects:* Move assigns the base and members of `*this` from the base and corresponding members of `rhs`.

2 *Returns:* `*this`.

```
void swap(basic_ifstream& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs` by calling `basic_istream<charT,traits>::swap(rhs)` and `sb.swap(rhs.sb)`.

```
template <class charT, class traits>
void swap(basic_ifstream<charT, traits>& x,
          basic_ifstream<charT, traits>& y);
```

4 *Effects:* `x.swap(y)`.

27.9.1.9 Member functions [ifstream.members]

```
basic_filebuf<charT,traits>* rdbuf() const;
```

1 *Returns:* `const_cast<basic_filebuf<charT,traits>*>(&sb)`.

```
bool is_open() const;
```

2 *Returns:* `rdbuf()->is_open()`.

```
void open(const char* s, ios_base::openmode mode = ios_base::in);
```

3 *Effects:* Calls `rdbuf()->open(s, mode | ios_base::in)`. If that function does not return a null pointer calls `clear()`, otherwise calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)).

```
void open(const string& s, ios_base::openmode mode = ios_base::in);
```

4 *Effects:* calls `open(s.c_str(), mode)`.

```
void close();
```

5 *Effects:* Calls `rdbuf()->close()` and, if that function returns a null pointer, calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)).

27.9.1.10 Class template `basic_ofstream` [ofstream]

```
namespace std {
  template <class charT, class traits = char_traits<charT> >
  class basic_ofstream : public basic_ostream<charT,traits> {
  public:
    typedef charT                    char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                   traits_type;

    // 27.9.1.11 Constructors:
    basic_ofstream();
    explicit basic_ofstream(const char* s,
        ios_base::openmode mode = ios_base::out);
    explicit basic_ofstream(const string& s,
        ios_base::openmode mode = ios_base::out);
    basic_ofstream(const basic_ofstream& rhs) = delete;
    basic_ofstream(basic_ofstream&& rhs);

    // 27.9.1.12 Assign/swap:
    basic_ofstream& operator=(const basic_ofstream& rhs) = delete;
    basic_ofstream& operator=(basic_ofstream&& rhs);
    void swap(basic_ofstream& rhs);

    // 27.9.1.13 Members:
    basic_filebuf<charT,traits>* rdbuf() const;

    bool is_open() const;
    void open(const char* s, ios_base::openmode mode = ios_base::out);
    void open(const string& s, ios_base::openmode mode = ios_base::out);
    void close();
  private:
    basic_filebuf<charT,traits> sb; // exposition only
  };

  template <class charT, class traits>
  void swap(basic_ofstream<charT, traits>& x,
            basic_ofstream<charT, traits>& y);
}
```

1 The class `basic_ofstream<charT, traits>` supports writing to named files. It uses a `basic_filebuf<charT, traits>` object to control the associated sequence. For the sake of exposition, the maintained data is presented here as:

— `sb`, the `filebuf` object.

27.9.1.11 `basic_ofstream` constructors [ofstream.cons]

```
basic_ofstream();
```

1 *Effects:* Constructs an object of class `basic_ofstream<charT,traits>`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_filebuf<charT,traits>()` (27.7.3.2, 27.9.1.2).

```
explicit basic_ofstream(const char* s,
    ios_base::openmode mode = ios_base::out);
```


2 *Effects:* Constructs an object of class `basic_ofstream<charT,traits>`, initializing the base class with `basic_ostream(&sb)` and initializing `sb` with `basic_filebuf<charT,traits>()` (27.7.3.2, 27.9.1.2), then calls `rdbuf()->open(s, mode|ios_base::out)`. If that function returns a null pointer, calls `setstate(failbit)`.

```
explicit basic_ofstream(const string& s,
    ios_base::openmode mode = ios_base::out);
```

3 *Effects:* the same as `basic_ofstream(s.c_str(), mode);`

```
basic_ofstream(basic_ofstream&& rhs);
```

4 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_filebuf`. Next `basic_ostream<charT,traits>::set_rdbuf(&sb)` is called to install the contained `basic_filebuf`.

27.9.1.12 Assign and swap [ofstream.assign]

```
basic_ofstream& operator=(basic_ofstream&& rhs);
```

1 *Effects:* Move assigns the base and members of `*this` from the base and corresponding members of `rhs`.

2 *Returns:* `*this`.

```
void swap(basic_ofstream& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs` by calling `basic_ostream<charT,traits>::swap(rhs)` and `sb.swap(rhs.sb)`.

```
template <class charT, class traits>
void swap(basic_ofstream<charT, traits>& x,
          basic_ofstream<charT, traits>& y);
```

4 *Effects:* `x.swap(y)`.

27.9.1.13 Member functions [ofstream.members]

```
basic_filebuf<charT,traits>* rdbuf() const;
```

1 *Returns:* `const_cast<basic_filebuf<charT,traits>*>(&sb)`.

```
bool is_open() const;
```

2 *Returns:* `rdbuf()->is_open()`.

```
void open(const char* s, ios_base::openmode mode = ios_base::out);
```

3 *Effects:* Calls `rdbuf()->open(s, mode | ios_base::out)`. If that function does not return a null pointer calls `clear()`, otherwise calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)).


```
void close();
```

4 *Effects:* Calls `rdbuf()->close()` and, if that function fails (returns a null pointer), calls `setstate(failbit)` (which may throw `ios_base::failure` (27.5.5.4)).

```
void open(const string& s, ios_base::openmode mode = ios_base::out);
```

5 *Effects:* calls `open(s.c_str(), mode);`

27.9.1.14 Class template `basic_fstream` [fstream]

```
namespace std {
  template <class charT, class traits=char_traits<charT> >
  class basic_fstream
    : public basic_iostream<charT,traits> {

  public:
    typedef charT                    char_type;
    typedef typename traits::int_type int_type;
    typedef typename traits::pos_type pos_type;
    typedef typename traits::off_type off_type;
    typedef traits                   traits_type;

    // constructors/destructor
    basic_fstream();
    explicit basic_fstream(const char* s,
        ios_base::openmode mode = ios_base::in|ios_base::out);
    explicit basic_fstream(const string& s,
        ios_base::openmode mode = ios_base::in|ios_base::out);
    basic_fstream(const basic_fstream& rhs) = delete;
    basic_fstream(basic_fstream&& rhs);

    // 27.9.1.16 Assign/swap:
    basic_fstream& operator=(const basic_fstream& rhs) = delete;
    basic_fstream& operator=(basic_fstream&& rhs);
    void swap(basic_fstream& rhs);

    // Members:
    basic_filebuf<charT,traits>* rdbuf() const;
    bool is_open() const;
    void open(const char* s,
        ios_base::openmode mode = ios_base::in|ios_base::out);
    void open(const string& s,
        ios_base::openmode mode = ios_base::in|ios_base::out);
    void close();

  private:
    basic_filebuf<charT,traits> sb; // exposition only
  };

  template <class charT, class traits>
  void swap(basic_fstream<charT, traits>& x,
            basic_fstream<charT, traits>& y);
}
```


1 The class template `basic_fstream<charT,traits>` supports reading and writing from named files. It uses a `basic_filebuf<charT,traits>` object to control the associated sequences. For the sake of exposition, the maintained data is presented here as:

— `sb`, the `basic_filebuf` object.

27.9.1.15 `basic_fstream` constructors [fstream.cons]

```
basic_fstream();
```

1 *Effects:* Constructs an object of class `basic_fstream<charT,traits>`, initializing the base class with `basic_iostream(&sb)` and initializing `sb` with `basic_filebuf<charT,traits>()`.

```
explicit basic_fstream(const char* s,
    ios_base::openmode mode = ios_base::in|ios_base::out);
```

2 *Effects:* Constructs an object of class `basic_fstream<charT, traits>`, initializing the base class with `basic_iostream(&sb)` and initializing `sb` with `basic_filebuf<charT, traits>()`. Then calls `rdbuf()->open(s, mode)`. If that function returns a null pointer, calls `setstate(failbit)`.

```
explicit basic_fstream(const string& s,
    ios_base::openmode mode = ios_base::in|ios_base::out);
```

3 *Effects:* the same as `basic_fstream(s.c_str(), mode);`

```
basic_fstream(basic_fstream&& rhs);
```

4 *Effects:* Move constructs from the rvalue `rhs`. This is accomplished by move constructing the base class, and the contained `basic_filebuf`. Next `basic_istream<charT,traits>::set_rdbuf(&sb)` is called to install the contained `basic_filebuf`.

27.9.1.16 Assign and swap [fstream.assign]

```
basic_fstream& operator=(basic_fstream&& rhs);
```

1 *Effects:* Move assigns the base and members of `*this` from the base and corresponding members of `rhs`.

2 *Returns:* `*this`.

```
void swap(basic_fstream& rhs);
```

3 *Effects:* Exchanges the state of `*this` and `rhs` by calling `basic_iostream<charT,traits>::swap(rhs)` and `sb.swap(rhs.sb)`.

```
template <class charT, class traits>
void swap(basic_fstream<charT, traits>& x,
          basic_fstream<charT, traits>& y);
```

4 *Effects:* `x.swap(y)`.

27.9.1.17 Member functions [fstream.members]

```
basic_filebuf<charT,traits>* rdbuf() const;
```

1 *Returns:* `const_cast<basic_filebuf<charT,traits>*>(&sb)`.

```
bool is_open() const;
```

2 *Returns:* `rdbuf()->is_open()`.

```
void open(const char* s,
    ios_base::openmode mode = ios_base::in|ios_base::out);
```

3 *Effects:* Calls `rdbuf()->open(s,mode)`. If that function does not return a null pointer calls `clear()`, otherwise calls `setstate(failbit)`, (which may throw `ios_base::failure`) (27.5.5.4).

```
void open(const string& s,
    ios_base::openmode mode = ios_base::in|ios_base::out);
```

4 *Effects:* calls `open(s.c_str(), mode);`

```
void close();
```

5 *Effects:* Calls `rdbuf()->close()` and, if that function returns returns a null pointer, calls `setstate(failbit)` (27.5.5.4) (which may throw `ios_base::failure`).

27.9.2 C library files [c.files]

1 Table 134 describes header `<cstdio>`. [*Note:* C++ does not define the function `gets`. — *end note*]

Table 134 — Header `<cstdio>` synopsis

Type	Name(s)				
Macros:					
BUFSIZ	FOPEN_MAX	SEEK_CUR	TMP_MAX	_IONBF	stdout
EOF	L_tmpnam	SEEK_END	_IOFBF	stderr	
FILENAME_MAX	NULL <cstdio>	SEEK_SET	_IOLBF	stdin	
Types:	FILE	fpos_t	size_t <cstdio>		
Functions:					
clearerr	fopen	fsetpos	putchar	snprintf	vscanf
fclose	fprintf	ftell	puts	sprintf	vsnprintf
feof	fputc	fwrite	remove	sscanf	vsprintf
ferror	fputs	getc	rename	tmpfile	vsscanf
fflush	fread	getchar	rewind	tmpnam	
fgetc	freopen	perror	scanf	ungetc	
fgetpos	fscanf	printf	setbuf	vfprintf	
fgets	fseek	putc	setvbuf	vprintf	

2 Calls to the function `tmpnam` with an argument of `NULL` may introduce a data race (17.6.5.9) with other calls to `tmpnam` with an argument of `NULL`.

See also: ISO C 7.9, Amendment 1 4.6.2.

3 Table 135 describes header `<cinttypes>`. [*Note:* The macros defined by `<cinttypes>` are provided unconditionally. In particular, the symbol `__STDC_FORMAT_MACROS`, mentioned in footnote 182 of the C standard, plays no role in C++. — *end note*]

Table 135 — Header `<cinttypes>` synopsis

Type	Name(s)
Macros:	
`PRI{d i o u x X}[FAST LEAST]{8 16 32 64}`	
`PRI{d i o u x X}{MAX PTR}`	
`SCN{d i o u x X}[FAST LEAST]{8 16 32 64}`	
`SCN{d i o u x X}{MAX PTR}`	
Types: `imaxdiv_t`	
Functions:	
`abs` `imaxabs` `strtoimax` `wcstoimax`	
`div` `imaxdiv` `strtoumax` `wcstoumax`	

4 The contents of header `<cinttypes>` are the same as the Standard C Library header `<inttypes.h>`, with the following changes:

— the header `<cinttypes>` includes the header `<cstdint>` instead of `<stdint.h>`, and

— if and only if the type `intmax_t` designates an extended integer type (3.9.1), the following function signatures are added:

```
intmax_t abs(intmax_t);
imaxdiv_t div(intmax_t, intmax_t);
```

which shall have the same semantics as the function signatures `intmax_t imaxabs(intmax_t)` and `imaxdiv_t imaxdiv(intmax_t, intmax_t)`, respectively.

